

# Package ‘mfrmr’

March 30, 2026

**Type** Package

**Title** Estimation and Diagnostics for Many-Facet Measurement Models

**Version** 0.1.4

**Description** Fits many-facet measurement models and returns diagnostics, reporting helpers, and reproducible analysis bundles using a native R implementation. Supports arbitrary facet counts, rating-scale and partial-credit parameterizations ('Andrich' (1978) <[doi:10.1007/BF02293814](https://doi.org/10.1007/BF02293814)>; 'Masters' (1982) <[doi:10.1007/BF02296272](https://doi.org/10.1007/BF02296272)>), marginal maximum likelihood estimation via the EM algorithm ('Bock' and 'Aitkin' (1981) <[doi:10.1007/BF02293801](https://doi.org/10.1007/BF02293801)>) and joint maximum likelihood estimation, plus tools for anchor review, interaction screening, linking workflows, and publication-ready summaries.

**URL** [https://ryuya-dot-com.github.io/R\\_package\\_mfrmr/](https://ryuya-dot-com.github.io/R_package_mfrmr/),  
[https://github.com/Ryuya-dot-com/R\\_package\\_mfrmr](https://github.com/Ryuya-dot-com/R_package_mfrmr)

**BugReports** [https://github.com/Ryuya-dot-com/R\\_package\\_mfrmr/issues](https://github.com/Ryuya-dot-com/R_package_mfrmr/issues)

**License** MIT + file LICENSE

**Language** en

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.3.3

**Depends** R (>= 4.1)

**Imports** dplyr, tidyr, tibble, purrr, stringr, psych, lifecycle, rlang,  
stats, utils

**Suggests** testthat (>= 3.0.0), covr, knitr, rmarkdown

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**NeedsCompilation** no

**Author** Ryuya Komuro [aut, cre] (ORCID:  
<<https://orcid.org/0000-0001-9205-0926>>)

**Maintainer** Ryuya Komuro <[ryuya.komuro.c4@tohoku.ac.jp](mailto:ryuya.komuro.c4@tohoku.ac.jp)>

Repository CRAN

Date/Publication 2026-03-30 19:20:02 UTC

## Contents

mfrmr-package . . . . .	4
analyze_dff . . . . .	10
analyze_facet_equivalence . . . . .	13
analyze_residual_pca . . . . .	16
anchor_to_baseline . . . . .	19
apa_table . . . . .	22
as.data.frame.mfrm_fit . . . . .	24
audit_mfrm_anchors . . . . .	25
bias_count_table . . . . .	28
bias_interaction_report . . . . .	30
bias_iteration_report . . . . .	32
bias_pairwise_report . . . . .	33
build_apa_outputs . . . . .	35
build_equating_chain . . . . .	37
build_fixed_reports . . . . .	40
build_mfrm_manifest . . . . .	42
build_mfrm_replay_script . . . . .	44
build_mfrm_sim_spec . . . . .	47
build_visual_summaries . . . . .	49
category_curves_report . . . . .	52
category_structure_report . . . . .	53
compare_mfrm . . . . .	55
compute_information . . . . .	58
data_quality_report . . . . .	60
describe_mfrm_data . . . . .	62
detect_anchor_drift . . . . .	64
diagnose_mfrm . . . . .	67
dif_interaction_table . . . . .	70
dif_report . . . . .	73
displacement_table . . . . .	74
ej2021_data . . . . .	76
estimate_all_bias . . . . .	78
estimate_bias . . . . .	80
estimation_iteration_report . . . . .	83
evaluate_mfrm_design . . . . .	84
evaluate_mfrm_signal_detection . . . . .	88
export_mfrm . . . . .	92
export_mfrm_bundle . . . . .	93
extract_mfrm_sim_spec . . . . .	96
facets_chisq_table . . . . .	98
facets_output_file_bundle . . . . .	100
facets_parity_report . . . . .	102

facet_quality_dashboard . . . . .	104
facet_statistics_report . . . . .	106
fair_average_table . . . . .	108
fit_mfrm . . . . .	110
interrater_agreement_table . . . . .	114
list_mfrmr_data . . . . .	117
load_mfrmr_data . . . . .	118
make_anchor_table . . . . .	119
measurable_summary_table . . . . .	120
mfrmr_compatibility_layer . . . . .	122
mfrmr_example_data . . . . .	124
mfrmr_linking_and_dff . . . . .	126
mfrmr_reporting_and_apa . . . . .	128
mfrmr_reports_and_tables . . . . .	130
mfrmr_visual_diagnostics . . . . .	132
mfrmr_workflow_methods . . . . .	135
mfrm_threshold_profiles . . . . .	137
plot.apa_table . . . . .	139
plot.mfrm_anchor_audit . . . . .	140
plot.mfrm_bundle . . . . .	142
plot.mfrm_data_description . . . . .	144
plot.mfrm_design_evaluation . . . . .	146
plot.mfrm_facets_run . . . . .	147
plot.mfrm_fit . . . . .	148
plot.mfrm_signal_detection . . . . .	151
plot_anchor_drift . . . . .	152
plot_bias_interaction . . . . .	154
plot_bubble . . . . .	157
plot_dif_heatmap . . . . .	159
plot_displacement . . . . .	160
plot_facets_chisq . . . . .	162
plot_facet_equivalence . . . . .	164
plot_facet_quality_dashboard . . . . .	166
plot_fair_average . . . . .	167
plot_information . . . . .	169
plot_interrater_agreement . . . . .	171
plot_qc_dashboard . . . . .	173
plot_qc_pipeline . . . . .	176
plot_residual_pca . . . . .	178
plot_unexpected . . . . .	180
plot_wright_unified . . . . .	182
precision_audit_report . . . . .	184
predict_mfrm_population . . . . .	186
predict_mfrm_units . . . . .	189
print.mfrm_apa_text . . . . .	191
rating_scale_table . . . . .	192
recommend_mfrm_design . . . . .	195
reference_case_audit . . . . .	196

reference_case_benchmark . . . . .	198
reporting_checklist . . . . .	200
run_mfrm_facets . . . . .	202
run_qc_pipeline . . . . .	205
sample_mfrm_plausible_values . . . . .	208
simulate_mfrm_data . . . . .	210
specifications_report . . . . .	213
subset_connectivity_report . . . . .	215
summary.apa_table . . . . .	216
summary.mfrm_anchor_audit . . . . .	217
summary.mfrm_apa_outputs . . . . .	218
summary.mfrm_bias . . . . .	220
summary.mfrm_bundle . . . . .	221
summary.mfrm_data_description . . . . .	223
summary.mfrm_design_evaluation . . . . .	224
summary.mfrm_diagnostics . . . . .	225
summary.mfrm_facets_run . . . . .	227
summary.mfrm_facet_dashboard . . . . .	228
summary.mfrm_fit . . . . .	229
summary.mfrm_plausible_values . . . . .	230
summary.mfrm_population_prediction . . . . .	231
summary.mfrm_signal_detection . . . . .	232
summary.mfrm_threshold_profiles . . . . .	233
summary.mfrm_unit_prediction . . . . .	235
unexpected_after_bias_table . . . . .	236
unexpected_response_table . . . . .	238

## Index 241

---

mfrmr-package	<i>mfrmr: Many-Facet Rasch Modeling in R</i>
---------------	--

---

## Description

mfrmr provides estimation, diagnostics, and reporting utilities for many-facet Rasch models (MFRM) using a native R implementation.

## Details

Recommended workflow:

1. Fit model with `fit_mfrm()`
2. Compute diagnostics with `diagnose_mfrm()`
3. Run residual PCA with `analyze_residual_pca()` if needed
4. Estimate interactions with `estimate_bias()`
5. Build narrative/report outputs with `build_apa_outputs()` and `build_visual_summaries()`

## Guide pages:

- [mfrmr\\_workflow\\_methods](#)
- [mfrmr\\_visual\\_diagnostics](#)
- [mfrmr\\_reports\\_and\\_tables](#)
- [mfrmr\\_reporting\\_and\\_apa](#)
- [mfrmr\\_linking\\_and\\_dff](#)
- [mfrmr\\_compatibility\\_layer](#)

## Companion vignettes:

- `vignette("mfrmr-workflow", package = "mfrmr")`
- `vignette("mfrmr-visual-diagnostics", package = "mfrmr")`
- `vignette("mfrmr-reporting-and-apa", package = "mfrmr")`
- `vignette("mfrmr-linking-and-dff", package = "mfrmr")`

## Function families:

- Model fitting: `fit_mfrm()`, `summary.mfrm_fit()`, `plot.mfrm_fit()`
- Legacy-compatible workflow wrapper: `run_mfrm_facets()`, `mfrmRFacets()`
- Diagnostics: `diagnose_mfrm()`, `summary(diag)`, `analyze_residual_pca()`, `plot_residual_pca()`
- Bias and interaction: `estimate_bias()`, `estimate_all_bias()`, `summary(bias)`, `bias_interaction_report()`, `plot_bias_interaction()`
- Differential functioning: `analyze_dff()`, `analyze_dif()`, `dif_interaction_table()`, `plot_dif_heatmap()`, `dif_report()`
- Design simulation: `build_mfrm_sim_spec()`, `extract_mfrm_sim_spec()`, `simulate_mfrm_data()`, `evaluate_mfrm_design()`, `evaluate_mfrm_signal_detection()`, `predict_mfrm_population()`, `predict_mfrm_units()`, `sample_mfrm_plausible_values()` (including fit-derived empirical / resampled / skeleton-based simulation specifications; fixed-calibration unit scoring currently requires `method = "MML"`)
- Reporting: `build_apa_outputs()`, `build_visual_summaries()`, `reporting_checklist()`, `apa_table()`
- Dashboards: `facet_quality_dashboard()`, `plot_facet_quality_dashboard()`
- Export / reproducibility: `build_mfrm_manifest()`, `build_mfrm_replay_script()`, `export_mfrm_bundle()`
- Equivalence: `analyze_facet_equivalence()`, `plot_facet_equivalence()`
- Data and anchors: `describe_mfrm_data()`, `audit_mfrm_anchors()`, `make_anchor_table()`, `load_mfrmr_data()`

## Data interface:

- Input analysis data is long format (one row per observed rating).
- Packaged simulation data is available via `load_mfrmr_data()` or `data()`.

## Interpreting output

Core object classes are:

- `mfrm_fit`: fitted model parameters and metadata.
- `mfrm_diagnostics`: fit/reliability/flag diagnostics.
- `mfrm_bias`: interaction bias estimates.
- `mfrm_dff` / `mfrm_dif`: differential-functioning contrasts and screening summaries.
- `mfrm_population_prediction`: scenario-level forecast summaries for one future design.
- `mfrm_unit_prediction`: fixed-calibration posterior summaries for future or partially observed persons.
- `mfrm_plausible_values`: approximate fixed-calibration posterior draws for future or partially observed persons.
- `mfrm_bundle` families: summary/report bundles and plotting payloads.

## Typical workflow

1. Prepare long-format data.
2. Fit with `fit_mfrm()`.
3. Diagnose with `diagnose_mfrm()`.
4. Run `analyze_dff()` or `estimate_bias()` when fairness or interaction questions matter.
5. Report with `build_apa_outputs()` and `build_visual_summaries()`.
6. For design planning, move to `build_mfrm_sim_spec()`, `evaluate_mfrm_design()`, and `predict_mfrm_population()`.
7. For future-unit scoring, refit or retain an MML calibration and then use `predict_mfrm_units()` or `sample_mfrm_plausible_values()`.

## Model formulation

The many-facet Rasch model (MFRM; Linacre, 1989) extends the basic Rasch model by incorporating multiple measurement facets into a single linear model on the log-odds scale.

### General MFRM equation

For an observation where person  $n$  with ability  $\theta_n$  is rated by rater  $j$  with severity  $\delta_j$  on criterion  $i$  with difficulty  $\beta_i$ , the probability of observing category  $k$  (out of  $K$  ordered categories) is:

$$P(X_{nij} = k \mid \theta_n, \delta_j, \beta_i, \tau) = \frac{\exp\left[\sum_{s=1}^k (\theta_n - \delta_j - \beta_i - \tau_s)\right]}{\sum_{c=0}^K \exp\left[\sum_{s=1}^c (\theta_n - \delta_j - \beta_i - \tau_s)\right]}$$

where  $\tau_s$  are the Rasch-Andrich threshold (step) parameters and  $\sum_{s=1}^0 (\cdot) \equiv 0$  by convention. Additional facets enter as additive terms in the linear predictor  $\eta = \theta_n - \delta_j - \beta_i - \dots$

This formulation generalises to any number of facets; the facets argument to `fit_mfrm()` accepts an arbitrary-length character vector.

### Rating Scale Model (RSM)

Under the RSM (Andrich, 1978), all levels of the step facet share a single set of threshold parameters  $\tau_1, \dots, \tau_K$ .

### Partial Credit Model (PCM)

Under the PCM (Masters, 1982), each level of the designated `step_facet` has its own threshold vector on the package's common observed score scale. In the current implementation, threshold locations may vary by step-facet level, but the fitted score range is still defined by one global category set taken from the observed data.

## Estimation methods

### Marginal Maximum Likelihood (MML)

MML integrates over the person ability distribution using Gauss-Hermite quadrature (Bock & Aitkin, 1981):

$$L = \prod_n \int P(\mathbf{X}_n | \theta, \boldsymbol{\delta}) \phi(\theta) d\theta \approx \prod_n \sum_{q=1}^Q w_q P(\mathbf{X}_n | \theta_q, \boldsymbol{\delta})$$

where  $\phi(\theta)$  is the assumed normal prior and  $(\theta_q, w_q)$  are quadrature nodes and weights. Person estimates are obtained post-hoc via Expected A Posteriori (EAP):

$$\hat{\theta}_n^{\text{EAP}} = \frac{\sum_q \theta_q w_q L(\mathbf{X}_n | \theta_q)}{\sum_q w_q L(\mathbf{X}_n | \theta_q)}$$

MML avoids the incidental-parameter problem and is generally preferred for smaller samples.

### Joint Maximum Likelihood (JML/JMLE)

JMLE estimates all person and facet parameters simultaneously as fixed effects by maximising the joint log-likelihood  $\ell(\boldsymbol{\theta}, \boldsymbol{\delta} | \mathbf{X})$  directly. It does not assume a parametric person distribution, which can be advantageous when the population shape is strongly non-normal, but parameter estimates are known to be biased when the number of persons is small relative to the number of items (Neyman & Scott, 1948).

See `fit_mfrm()` for practical guidance on choosing between the two.

## Statistical background

Key statistics reported throughout the package:

### Infit (Information-Weighted Mean Square)

Weighted average of squared standardized residuals, where weights are the model-based variance of each observation:

$$\text{Infit}_j = \frac{\sum_i Z_{ij}^2 \text{Var}_i w_i}{\sum_i \text{Var}_i w_i}$$

Expected value is 1.0 under model fit. Values below 0.5 suggest overfit (Mead-style responses); values above 1.5 suggest underfit (noise or misfit). Infit is most sensitive to unexpected patterns among on-target observations (Wright & Masters, 1982).

Note: The 0.5–1.5 range is a widely used rule of thumb (Bond & Fox, 2015). Acceptable ranges may differ by context: 0.6–1.4 for high-stakes testing; 0.7–1.3 for clinical instruments; up to 0.5–1.7 for surveys and exploratory work (Linacre, 2002).

### **Outfit (Unweighted Mean Square)**

Simple average of squared standardized residuals:

$$\text{Outfit}_j = \frac{\sum_i Z_{ij}^2 w_i}{\sum_i w_i}$$

Same expected value and flagging thresholds as Infit, but more sensitive to extreme off-target outliers (e.g., a high-ability person scoring the lowest category).

### **ZSTD (Standardized Fit Statistic)**

Wilson-Hilferty cube-root transformation that converts the mean-square chi-square ratio to an approximate standard normal deviate:

$$\text{ZSTD} = \frac{\text{MnSq}^{1/3} - (1 - 2/(9 df))}{\sqrt{2/(9 df)}}$$

Values near 0 indicate expected fit;  $|\text{ZSTD}| > 2$  flags potential misfit at the 5% level. Infit and Outfit value.

### **PTMEA (Point-Measure Correlation)**

Pearson correlation between observed scores and estimated person measures within each facet level. Positive values indicate that scoring aligns with the latent trait dimension; negative values suggest reversed orientation or scoring errors.

### **Separation**

Package-reported separation is the ratio of adjusted true standard deviation to root-mean-square measurement error:

$$G = \frac{\text{SD}_{\text{adj}}}{\text{RMSE}}$$

where  $\text{SD}_{\text{adj}} = \sqrt{\text{ObservedVariance} - \text{ErrorVariance}}$ . Higher values indicate the facet discriminates more statistically distinct levels along the measured variable. In `mfrmr`, `Separation` is the model-based value and `RealSeparation` provides a more conservative companion based on `RealSE`.

### **Reliability**

$$R = \frac{G^2}{1 + G^2}$$

Analogous to Cronbach's alpha or KR-20 for the reproducibility of element ordering. In `mfrmr`, `Reliability` is the model-based value and `RealReliability` gives the conservative companion based on `RealSE`. For MML, these are anchored to observed-information `ModelSE` estimates for non-person facets; JML keeps them as exploratory summaries.

### **Strata**

Number of statistically distinguishable groups of elements:

$$H = \frac{4G + 1}{3}$$

Three or more strata are commonly used as a practical target (Wright & Masters, 1982), but in this package the estimate inherits the same approximation limits as the separation index.

### Key references

- Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika*, 43, 561–573.
- Bond, T. G., & Fox, C. M. (2015). *Applying the Rasch model* (3rd ed.). Routledge.
- Linacre, J. M. (1989). *Many-facet Rasch measurement*. MESA Press.
- Linacre, J. M. (2002). What do Infit and Outfit, mean-square and standardized mean? *Rasch Measurement Transactions*, 16(2), 878.
- Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika*, 47, 149–174.
- Wright, B. D., & Masters, G. N. (1982). *Rating scale analysis*. MESA Press.
- Wright, B. D., & Linacre, J. M. (1994). Reasonable mean-square fit values. *Rasch Measurement Transactions*, 8(3), 370.

### Model selection

#### RSM vs PCM

The Rating Scale Model (RSM; Andrich, 1978) assumes all levels of the step facet share identical threshold parameters. The Partial Credit Model (PCM; Masters, 1982) allows each level of the `step_facet` to have its own set of thresholds on the package’s shared observed score scale. Use RSM when the rating rubric is identical across all items/criteria; use PCM when category boundaries are expected to vary by item or criterion. In the current implementation, PCM still assumes one common observed score support across the fitted data, so it should not be described as a fully mixed-category model with arbitrary item-specific category counts.

#### MML vs JML

Marginal Maximum Likelihood (MML) integrates over the person ability distribution using Gauss-Hermite quadrature and does not directly estimate person parameters; person estimates are computed post-hoc via Expected A Posteriori (EAP). Joint Maximum Likelihood (JML/JMLE) estimates all person and facet parameters simultaneously as fixed effects.

MML is generally preferred for smaller samples because it avoids the incidental-parameter problem of JML. JML is faster and does not assume a normal person distribution, which may be an advantage when the population shape is strongly non-normal.

See `fit_mfrm()` for usage.

### Author(s)

**Maintainer:** Ryuya Komuro <ryuya.komuro.c4@tohoku.ac.jp> ([ORCID](#))

**See Also**

Useful links:

- [https://ryuya-dot-com.github.io/R\\_package\\_mfrmr/](https://ryuya-dot-com.github.io/R_package_mfrmr/)
- [https://github.com/Ryuya-dot-com/R\\_package\\_mfrmr](https://github.com/Ryuya-dot-com/R_package_mfrmr)
- Report bugs at [https://github.com/Ryuya-dot-com/R\\_package\\_mfrmr/issues](https://github.com/Ryuya-dot-com/R_package_mfrmr/issues)

**Examples**

```
mfrm_threshold_profiles()
list_mfrmr_data()
```

---

analyze\_dff

*Differential facet functioning analysis*

---

**Description**

Tests whether the difficulty of facet levels differs across a grouping variable (e.g., whether rater severity differs for male vs. female examinees, or whether item difficulty differs across rater sub-groups).

analyze\_dif() is retained for compatibility with earlier package versions. In many-facet workflows, prefer [analyze\\_dff\(\)](#) as the primary entry point.

**Usage**

```
analyze_dff(
  fit,
  diagnostics,
  facet,
  group,
  data = NULL,
  focal = NULL,
  method = c("residual", "refit"),
  min_obs = 10,
  p_adjust = "holm"
)

analyze_dif(...)
```

**Arguments**

fit	Output from <a href="#">fit_mfrmr()</a> .
diagnostics	Output from <a href="#">diagnose_mfrmr()</a> .
facet	Character scalar naming the facet whose elements are tested for differential functioning (for example, "Criterion" or "Rater").

group	Character scalar naming the column in the data that defines the grouping variable (e.g., "Gender", "Site").
data	Optional data frame containing at least the group column and the same person/facet/score columns used to fit the model. If NULL (default), the data stored in fit\$prep\$data is used.
focal	Optional character vector of group levels to treat as focal. If NULL (default), all pairwise group comparisons are performed.
method	Analysis method: "residual" (default) uses the fitted model's residuals without re-estimation; "refit" re-estimates the model within each group subset. The residual method is faster and avoids convergence issues with small subsets.
min_obs	Minimum number of observations per cell (facet-level x group). Cells below this threshold are flagged as sparse and their statistics set to NA. Default 10.
p_adjust	Method for multiple-comparison adjustment, passed to <code>stats::p.adjust()</code> . Default is "holm".
...	Passed directly to <code>analyze_dff()</code> .

## Details

**Differential facet functioning (DFF)** occurs when the difficulty or severity of a facet element differs across subgroups of the population, after controlling for overall ability. In an MFRM context this generalises classical DIF (which applies to items) to any facet: raters, criteria, tasks, etc.

Differential functioning is a threat to measurement fairness: if Criterion 1 is harder for Group A than Group B at the same ability level, the measurement scale is no longer group-invariant.

Two methods are available:

**Residual method** (`method = "residual"`): Uses the existing fitted model's observation-level residuals. For each facet-level  $\times$  group cell, the observed and expected score sums are aggregated and a standardized residual is computed as:

$$z = \frac{\sum(X_{obs} - E_{exp})}{\sqrt{\sum \text{Var}}}$$

Pairwise contrasts between groups compare the mean observed-minus-expected difference for each facet level, with uncertainty summarized by a Welch/Satterthwaite approximation. This method is fast, stable with small subsets, and does not require re-estimation. Because the resulting contrast is not a logit-scale parameter difference, the residual method is treated as a screening procedure rather than an ETS-style classifier.

**Refit method** (`method = "refit"`): Subsets the data by group, refits the MFRM model within each subset, anchors all non-target facets back to the baseline calibration when possible, and compares the resulting facet-level estimates using a Welch t-statistic:

$$t = \frac{\hat{\delta}_1 - \hat{\delta}_2}{\sqrt{SE_1^2 + SE_2^2}}$$

This provides group-specific parameter estimates on a common scale when linking anchors are available, but is slower and may encounter convergence issues with small subsets. ETS categories are reported only for contrasts whose subgroup calibrations retained enough linking anchors to

support a common-scale interpretation and whose subgroup precision remained on the package's model-based MML path.

When facet refers to an item-like facet (for example `Criterion`), this recovers the familiar DIF case. When facet refers to raters or prompts/tasks, the same machinery supports DRF/DPF-style analyses.

For the refit method only, effect size is classified following the ETS (Educational Testing Service) DIF guidelines when subgroup calibrations are both linked and eligible for model-based inference:

- **A (Negligible)**:  $|\Delta| < 0.43$  logits
- **B (Moderate)**:  $0.43 \leq |\Delta| < 0.64$  logits
- **C (Large)**:  $|\Delta| \geq 0.64$  logits

Multiple comparisons are adjusted using Holm's step-down procedure by default, which controls the family-wise error rate without assuming independence. Alternative methods (e.g., "BH" for false discovery rate) can be specified via `p_adjust`.

### Value

An object of class `mfrm_dff` (with compatibility class `mfrm_dif`) with:

- `dif_table`: data.frame of differential-functioning contrasts.
- `cell_table`: (residual method) per-cell detail table.
- `summary`: counts by screening or ETS classification.
- `group_fits`: (refit method) per-group facet estimates.
- `config`: list with facet, group, method, min\_obs, p\_adjust settings.

### Choosing a method

In most first-pass DFF screening, start with `method = "residual"`. It is faster, reuses the fitted model, and is less fragile in smaller subsets. Use `method = "refit"` when you specifically want group-specific parameter estimates and can tolerate extra computation. Both methods should yield similar conclusions when sample sizes are adequate ( $N \geq 100$  per group is a useful guideline for stable differential-functioning detection).

### Interpreting output

- `$dif_table`: one row per facet-level x group-pair with contrast, SE, t-statistic, p-value, adjusted p-value, effect metric, and method-appropriate classification. Includes `Method`, `N_Group1`, `N_Group2`, `EffectMetric`, `ClassificationSystem`, `ContrastBasis`, `SEBasis`, `StatisticLabel`, `ProbabilityMetric`, `DFBasis`, `ReportingUse`, `PrimaryReportingEligible`, and sparse columns.
- `$cell_table`: (residual method only) per-cell detail with `N`, `ObsScore`, `ExpScore`, `ObsExpAvg`, `StdResidual`.
- `$summary`: counts by screening result (`method = "residual"`) or ETS category plus linked-screening and insufficient-linking rows (`method = "refit"`).
- `$group_fits`: (refit method only) list of per-group facet estimates and subgroup linking diagnostics.

**Typical workflow**

1. Fit a model with `fit_mfrm()`.
2. Run `analyze_dff(fit, diagnostics, facet = "Criterion", group = "Gender", data = my_data)`.
3. Inspect `$dif_table` for flagged levels and `$summary` for counts.
4. Use `dif_interaction_table()` when you need cell-level diagnostics.
5. Use `plot_dif_heatmap()` or `dif_report()` for communication.

**See Also**

`fit_mfrm()`, `estimate_bias()`, `compare_mfrm()`, `dif_interaction_table()`, `plot_dif_heatmap()`, `dif_report()`, `subset_connectivity_report()`, `mfrmr_linking_and_dff`

**Examples**

```
toy <- load_mfrmr_data("example_bias")

fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score",
               method = "JML", model = "RSM", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
dff <- analyze_dff(fit, diag, facet = "Rater", group = "Group", data = toy)
dff$summary
head(dff$dif_table[, c("Level", "Group1", "Group2", "Contrast", "Classification")])
sc <- subset_connectivity_report(fit, diagnostics = diag)
plot(sc, type = "design_matrix", draw = FALSE)
if ("ScaleLinkStatus" %in% names(dff$dif_table)) {
  unique(dff$dif_table$ScaleLinkStatus)
}
```

---

analyze\_facet\_equivalence

*Analyze practical equivalence within a facet*

---

**Description**

Analyze practical equivalence within a facet

**Usage**

```
analyze_facet_equivalence(
  fit,
  diagnostics = NULL,
  facet = NULL,
  equivalence_bound = 0.5,
  conf_level = 0.95
)
```

**Arguments**

fit	Output from <code>fit_mfrm()</code> .
diagnostics	Optional output from <code>diagnose_mfrm()</code> . When NULL, diagnostics are computed with <code>residual_pca = "none"</code> .
facet	Character scalar naming the non-person facet to evaluate. If NULL, the function prefers a rater-like facet and otherwise uses the first model facet.
equivalence_bound	Practical-equivalence bound in logits. Default 0.5.
conf_level	Confidence level used for the forest-style interval view. Default 0.95.

**Details**

This function tests whether facet elements (e.g., raters) are similar enough to be treated as practically interchangeable, rather than merely testing whether they differ significantly. This is the key distinction from a standard chi-square heterogeneity test: absence of evidence for difference is not evidence of equivalence.

The function uses existing facet estimates and their standard errors from `diagnostics$measures`; no re-estimation is performed.

The bundle combines four complementary views:

1. **Fixed chi-square test:** tests  $H_0$ : all element measures are equal. A non-significant result is *necessary but not sufficient* for interchangeability. It is reported as context, not as direct evidence of equivalence.
2. **Pairwise TOST (Two One-Sided Tests):** for each pair of elements, tests whether the difference falls within  $\pm$ equivalence\_bound. The TOST procedure (Schuirmann, 1987) rejects the null hypothesis of *non-equivalence* when both one-sided tests are significant at level  $\alpha$ . A pair is declared "Equivalent" when the TOST p-value  $< 0.05$ .
3. **BIC-based Bayes-factor heuristic:** an approximate screening tool (not full Bayesian inference) that compares the evidence for a common-facet model (all elements equal) against a heterogeneity model (elements differ). Values  $> 3$  favour the common-facet model;  $< 1/3$  favour heterogeneity.
4. **ROPE-style grand-mean proximity:** the proportion of each element's normal-approximation confidence distribution that falls within  $\pm$ equivalence\_bound of the weighted grand mean. This is a descriptive proximity summary, not a Bayesian ROPE decision rule around a pre-specified null value.

**Choosing** equivalence\_bound: the default of 0.5 logits is a moderate criterion. For high-stakes certification, 0.3 logits may be appropriate; for exploratory or low-stakes contexts, 1.0 logits may suffice. The bound should reflect the smallest difference that would be practically meaningful in your application.

**Value**

A named list with class `mfrm_facet_equivalence`.

**What this analysis means**

analyze\_facet\_equivalence() is a practical-interchangeability screen. It asks whether facet levels are close enough, under a user-defined logit bound, to be treated as practically similar for the current use case.

**What this analysis does not justify**

- A non-significant chi-square result is not evidence of equivalence.
- Forest/ROPE displays are descriptive and do not replace the pairwise TOST decision rule.
- The BIC-based Bayes-factor summary is a heuristic screen, not a full Bayesian equivalence analysis.

**Interpreting output**

Start with summary\$Decision, which is a conservative summary of the pairwise TOST results. Then use the remaining tables as context:

- chi\_square: is there broad heterogeneity in the facet?
- pairwise: which specific pairs meet the practical-equivalence bound?
- rope / forest: how close is each level to the facet grand mean?

Smaller equivalence\_bound values make the criterion stricter. If the decision is "partial\_pairwise\_equivalence", that means some pairwise contrasts satisfy the practical-equivalence bound but not all of them do.

**Decision rule**

The final Decision is a pairwise TOST summary rather than a global equivalence proof. If all pairwise contrasts satisfy the practical-equivalence bound, the facet is labeled "all\_pairs\_equivalent".

If at least one, but not all, pairwise contrasts are equivalent, the facet is labeled "partial\_pairwise\_equivalence".

If no pairwise contrasts meet the practical-equivalence bound, the facet is labeled "no\_pairwise\_equivalence\_established". The chi-square, Bayes-factor, and grand-mean proximity summaries are reported as descriptive context.

**How to read the main outputs**

- summary: one-row pairwise-TOST decision summary and aggregate context.
- pairwise: pair-level TOST detail; use this for the primary inferential read.
- chi\_square: broad heterogeneity screen.
- rope / forest: level-wise proximity to the weighted grand mean.

**Recommended next step**

If the result is borderline or high-stakes, re-run the analysis with a tighter or looser equivalence\_bound, then inspect pairwise and plot\_facet\_equivalence() before deciding how strongly to claim interchangeability.

**Typical workflow**

1. Fit a model with `fit_mfrm()`.
2. Run `analyze_facet_equivalence()` for the facet you want to screen.
3. Read summary and `chi_square` first.
4. Use `plot_facet_equivalence()` to inspect which levels drive the result.

**Output**

The returned bundle has class `mfrm_facet_equivalence` and includes:

- `summary`: one-row overview with convergent decision
- `chi_square`: fixed chi-square / separation summary
- `pairwise`: pairwise TOST detail table
- `rope`: element-wise ROPE probabilities around the weighted grand mean
- `forest`: element-wise estimate, confidence interval, and ROPE status
- `settings`: applied facet and threshold settings

**See Also**

`facets_chisq_table()`, `fair_average_table()`, `plot_facet_equivalence()`

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score",
               method = "JML", maxit = 25)
eq <- analyze_facet_equivalence(fit, facet = "Rater")
eq$summary[, c("Facet", "Elements", "Decision", "MeanROPE")]
head(eq$pairwise[, c("ElementA", "ElementB", "Equivalent")])
```

---

`analyze_residual_pca` *Run exploratory residual PCA summaries*

---

**Description**

Legacy-compatible residual diagnostics can be inspected in two ways:

1. overall residual PCA on the person x combined-facet matrix
2. facet-specific residual PCA on person x facet-level matrices

**Usage**

```
analyze_residual_pca(
  diagnostics,
  mode = c("overall", "facet", "both"),
  facets = NULL,
  pca_max_factors = 10L
)
```

## Arguments

diagnostics	Output from <code>diagnose_mfrm()</code> or <code>fit_mfrm()</code> .
mode	"overall", "facet", or "both".
facets	Optional subset of facets for facet-specific PCA.
pca_max_factors	Maximum number of retained components.

## Details

The function works on standardized residual structures derived from `diagnose_mfrm()`. When a fitted object from `fit_mfrm()` is supplied, diagnostics are computed internally.

Conceptually, this follows the Rasch residual-PCA tradition of examining structure in model residuals after the primary Rasch dimension has been extracted. In `mfrm`, however, the implementation is an **exploratory many-facet adaptation**: it works on standardized residual matrices built as person x combined-facet or person x facet-level layouts, rather than reproducing FACETS/Winsteps residual-contrast tables one-to-one.

Output tables use:

- Component: principal-component index (1, 2, ...)
- Eigenvalue: eigenvalue for each component
- Proportion: component variance proportion
- Cumulative: cumulative variance proportion

For mode = "facet" or "both", `by_facet_table` additionally includes a Facet column.

`summary(pca)` is supported through `summary()`. `plot(pca)` is dispatched through `plot()` for class `mfrm_residual_pca`. Available types include "overall\_sceer", "facet\_sceer", "overall\_loadings", and "facet\_loadings".

## Value

A named list with:

- mode: resolved mode used for computation
- facet\_names: facets analyzed
- overall: overall PCA bundle (or NULL)
- by\_facet: named list of facet PCA bundles
- overall\_table: variance table for overall PCA
- by\_facet\_table: stacked variance table across facets

## Interpreting output

Use `overall_table` first:

- early components with noticeably larger eigenvalues or proportions suggest stronger residual structure that may deserve follow-up.

Then inspect by `by_facet_table`:

- helps localize which facet contributes most to residual structure.

Finally, inspect loadings via `plot_residual_pca()` to identify which variables/elements drive each component.

## References

The residual-PCA idea follows the Rasch residual-structure literature, especially Linacre's discussions of principal components of Rasch residuals. The current `mfrmr` implementation should be interpreted as an exploratory extension for many-facet workflows rather than as a direct reproduction of a single FACETS/Winsteps output table.

- Linacre, J. M. (1998). *Structure in Rasch residuals: Why principal components analysis (PCA)?* Rasch Measurement Transactions, 12(2), 636.
- Linacre, J. M. (1998). *Detecting multidimensionality: Which residual data-type works best?* Journal of Outcome Measurement, 2(3), 266-283.

## Typical workflow

1. Fit model and run `diagnose_mfrmr()` with `residual_pca = "none"` or `"both"`.
2. Call `analyze_residual_pca(..., mode = "both")`.
3. Review `summary(pca)`, then plot scree/loadings.
4. Cross-check with fit/misfit diagnostics before conclusions.

## See Also

[diagnose\\_mfrmr\(\)](#), [plot\\_residual\\_pca\(\)](#), [mfrmr\\_visual\\_diagnostics](#)

## Examples

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrmr(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrmr(fit, residual_pca = "both")
pca <- analyze_residual_pca(diag, mode = "both")
pca2 <- analyze_residual_pca(fit, mode = "both")
summary(pca)
p <- plot_residual_pca(pca, mode = "overall", plot_type = "scree", draw = FALSE)
class(p)
head(p$data)
head(pca$overall_table)
```

---

anchor\_to\_baseline      *Fit new data anchored to a baseline calibration*

---

### Description

Re-estimates a many-facet Rasch model on new data while holding selected facet parameters fixed at the values from a previous (baseline) calibration. This is the standard workflow for placing new data onto an existing scale, linking test forms, or carrying a baseline calibration across administration windows.

### Usage

```
anchor_to_baseline(
  new_data,
  baseline_fit,
  person,
  facets,
  score,
  anchor_facets = NULL,
  include_person = FALSE,
  weight = NULL,
  model = NULL,
  method = NULL,
  anchor_policy = "warn",
  ...
)

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

## S3 method for class 'mfrm_anchored_fit'
summary(object, ...)

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

### Arguments

<code>new_data</code>	Data frame in long format (one row per rating).
<code>baseline_fit</code>	An <code>mfrm_fit</code> object from a previous calibration.
<code>person</code>	Character column name for person/examinee.
<code>facets</code>	Character vector of facet column names.
<code>score</code>	Character column name for the rating score.
<code>anchor_facets</code>	Character vector of facets to anchor (default: all non-Person facets).
<code>include_person</code>	If TRUE, also anchor person estimates.

weight	Optional character column name for observation weights.
model	Scale model override; defaults to baseline model.
method	Estimation method override; defaults to baseline method.
anchor_policy	How to handle anchor issues: "warn", "error", "silent".
...	Ignored.
x	An mfrm_anchored_fit object.
object	An mfrm_anchored_fit object (for summary).

## Details

This function automates the baseline-anchored calibration workflow:

1. Extracts anchor values from the baseline fit using `make_anchor_table()`.
2. Re-estimates the model on `new_data` with those anchors fixed via `fit_mfrm(..., anchors = anchor_table)`.
3. Runs `diagnose_mfrm()` on the anchored fit.
4. Computes element-level differences (new estimate minus baseline estimate) for every common element.

The `model` and `method` arguments default to the baseline fit's settings so the calibration framework remains consistent. Elements present in the anchor table but absent from the new data are handled according to `anchor_policy`: "warn" (default) emits a message, "error" stops execution, and "silent" ignores silently.

The returned drift table is best interpreted as an anchored consistency check. When a facet is fixed through `anchor_facets`, those anchored levels are constrained in the new run, so their reported differences are not an independent drift analysis. For genuine cross-wave drift monitoring, fit the waves separately and use `detect_anchor_drift()` on the resulting fits.

Element-level differences are calculated for every element that appears in both the baseline and the new calibration:

$$\Delta_e = \hat{\delta}_{e,\text{new}} - \hat{\delta}_{e,\text{base}}$$

An element is **flagged** when  $|\Delta_e| > 0.5$  logits or  $|\Delta_e/SE_{\Delta_e}| > 2.0$ , where  $SE_{\Delta_e} = \sqrt{SE_{\text{base}}^2 + SE_{\text{new}}^2}$ .

## Value

Object of class `mfrm_anchored_fit` with components:

**fit** The anchored `mfrm_fit` object.

**diagnostics** Output of `diagnose_mfrm()` on the anchored fit.

**baseline\_anchors** Anchor table extracted from the baseline.

**drift** Tibble of element-level drift statistics.

### Which function should I use?

- Use `anchor_to_baseline()` when you have one new dataset and want to place it directly on a baseline scale.
- Use `detect_anchor_drift()` when you already have multiple fitted waves and want to compare their stability.
- Use `build_equating_chain()` when you need cumulative offsets across an ordered series of waves.

### Interpreting output

- `$drift`: one row per common element with columns Facet, Level, Baseline, New, Drift, SE\_Baseline, SE\_New, SE\_Diff, Drift\_SE\_Ratio, and Flag. Read this as an anchored consistency table. Small absolute differences indicate that the anchored re-fit stayed close to the baseline scale. Flagged rows warrant review, but they are not a substitute for a separate drift study on unanchored common elements.
- `$fit`: the full anchored `mfrm_fit` object, usable with `diagnose_mfrm()`, `measurable_summary_table()`, etc.
- `$diagnostics`: pre-computed diagnostics for the anchored calibration.
- `$baseline_anchors`: the anchor table fed to `fit_mfrm()`, useful for auditing which elements were constrained.

### Typical workflow

1. Fit the baseline model: `fit1 <- fit_mfrm(...)`.
2. Collect new data (e.g., a later administration).
3. Call `res <- anchor_to_baseline(new_data, fit1, ...)`.
4. Inspect `summary(res)` to confirm the anchored run remains close to the baseline scale.
5. For multi-wave drift monitoring, fit waves separately and pass the fits to `detect_anchor_drift()` or `build_equating_chain()`.

### See Also

`fit_mfrm()`, `make_anchor_table()`, `detect_anchor_drift()`, `diagnose_mfrm()`, `build_equating_chain()`, `mfrmr_linking_and_dff`

### Examples

```
d1 <- load_mfrmr_data("study1")
keep1 <- unique(d1$Person)[1:15]
d1 <- d1[d1$Person %in% keep1, , drop = FALSE]
fit1 <- fit_mfrm(d1, "Person", c("Rater", "Criterion"), "Score",
               method = "JML", maxit = 15)
d2 <- load_mfrmr_data("study2")
keep2 <- unique(d2$Person)[1:15]
d2 <- d2[d2$Person %in% keep2, , drop = FALSE]
res <- anchor_to_baseline(d2, fit1, "Person",
                       c("Rater", "Criterion"), "Score",
                       anchor_facets = "Criterion")
```

```
summary(res)
head(res$drift[, c("Facet", "Level", "Drift", "Flag")])
res$baseline_anchors[1:3, ]
```

---

apa\_table

*Build APA-style table output using base R structures*


---

## Description

Build APA-style table output using base R structures

## Usage

```
apa_table(
  x,
  which = NULL,
  diagnostics = NULL,
  digits = 2,
  caption = NULL,
  note = NULL,
  bias_results = NULL,
  context = list(),
  whexact = FALSE,
  branch = c("apa", "facets")
)
```

## Arguments

x	A data.frame, mfrm_fit, diagnostics list, or bias-result list.
which	Optional table selector when x has multiple tables.
diagnostics	Optional diagnostics from <a href="#">diagnose_mfrm()</a> (used when x is mfrm_fit and which targets diagnostics tables).
digits	Number of rounding digits for numeric columns.
caption	Optional caption text.
note	Optional note text.
bias_results	Optional output from <a href="#">estimate_bias()</a> used when auto-generating APA metadata for fit-based tables.
context	Optional context list forwarded when auto-generating APA metadata for fit-based tables.
whexact	Logical forwarded to APA metadata helpers.
branch	Output branch: "apa" for manuscript-oriented labels, "facets" for FACETS-aligned labels.

**Details**

This helper avoids styling dependencies and returns a reproducible base data.frame plus metadata.

Supported which values:

- For mfrm\_fit: "summary", "person", "facets", "steps"
- For diagnostics list: "overall\_fit", "measures", "fit", "reliability", "facets\_chisq", "bias", "interactions", "interrater\_summary", "interrater\_pairs", "obs"
- For bias-result list: "table", "summary", "chi\_sq"

**Value**

A list of class apa\_table with fields:

- table (data.frame)
- which
- caption
- note
- digits
- branch, style

**Interpreting output**

- table: plain data.frame ready for export or further formatting.
- which: source component that produced the table.
- caption/note: manuscript-oriented metadata stored with the table.

**Typical workflow**

1. Build table object with `apa_table(...)`.
2. Inspect quickly with `summary(tbl)`.
3. Render base preview via `plot(tbl, ...)` or export `tbl$table`.

**See Also**

[fit\\_mfrm\(\)](#), [diagnose\\_mfrm\(\)](#), [build\\_apa\\_outputs\(\)](#), [reporting\\_checklist\(\)](#), [mfrm\\_reporting\\_and\\_apa](#)

**Examples**

```
toy <- load_mfrm_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
tbl <- apa_table(fit, which = "summary", caption = "Model summary", note = "Toy example")
tbl_facets <- apa_table(fit, which = "summary", branch = "facets")
summary(tbl)
p <- plot(tbl, draw = FALSE)
p_facets <- plot(tbl_facets, type = "numeric_profile", draw = FALSE)
if (interactive()) {
  plot(
```

```
tbl,
  type = "numeric_profile",
  main = "APA Table Numeric Profile (Customized)",
  palette = c(numeric_profile = "#2b8cbe", grid = "#d9d9d9"),
  label_angle = 45
)
}
class(tbl)
tbl$note
```

---

```
as.data.frame.mfrm_fit
```

*Convert mfrm\_fit to a tidy data.frame*

---

### Description

Returns all facet-level estimates (person and others) in a single tidy data.frame. Useful for quick interactive export: `write.csv(as.data.frame(fit), "results.csv")`.

### Usage

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

### Arguments

<code>x</code>	An <code>mfrm_fit</code> object from <code>fit_mfrm</code> .
<code>row.names</code>	Ignored (included for S3 generic compatibility).
<code>optional</code>	Ignored (included for S3 generic compatibility).
<code>...</code>	Additional arguments (ignored).

### Details

This method is intentionally lightweight: it returns just three columns (Facet, Level, Estimate) so that the result is easy to inspect, join, or write to disk.

### Value

A data.frame with columns Facet, Level, Estimate.

### Interpreting output

Person estimates are returned with Facet = "Person". All non-person facets are stacked underneath in the same schema.

### Typical workflow

1. Fit a model with `fit_mfrm()`.
2. Convert with `as.data.frame(fit)` for a compact long-format export.
3. Join additional diagnostics later if you need SE or fit statistics.

### See Also

`fit_mfrm`, `export_mfrm`

### Examples

```
toy <- load_mfrm_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score",
               method = "JML", model = "RSM", maxit = 25)
head(as.data.frame(fit))
```

---

audit\_mfrm\_anchors      *Audit and normalize anchor/group-anchor tables*

---

### Description

Audit and normalize anchor/group-anchor tables

### Usage

```
audit_mfrm_anchors(  
  data,  
  person,  
  facets,  
  score,  
  anchors = NULL,  
  group_anchors = NULL,  
  weight = NULL,  
  rating_min = NULL,  
  rating_max = NULL,  
  keep_original = FALSE,  
  min_common_anchors = 5L,  
  min_obs_per_element = 30,  
  min_obs_per_category = 10,  
  noncenter_facet = "Person",  
  dummy_facets = NULL  
)
```

**Arguments**

<code>data</code>	A data.frame in long format (one row per rating event).
<code>person</code>	Column name for person IDs.
<code>facets</code>	Character vector of facet column names.
<code>score</code>	Column name for observed score.
<code>anchors</code>	Optional anchor table (Facet, Level, Anchor).
<code>group_anchors</code>	Optional group-anchor table (Facet, Level, Group, GroupValue).
<code>weight</code>	Optional weight/frequency column name.
<code>rating_min</code>	Optional minimum category value.
<code>rating_max</code>	Optional maximum category value.
<code>keep_original</code>	Keep original category values.
<code>min_common_anchors</code>	Minimum anchored levels per linking facet used in recommendations (default 5).
<code>min_obs_per_element</code>	Minimum weighted observations per facet level used in recommendations (default 30).
<code>min_obs_per_category</code>	Minimum weighted observations per score category used in recommendations (default 10).
<code>noncenter_facet</code>	One facet to leave non-centered.
<code>dummy_facets</code>	Facets to fix at zero.

**Details**

**Anchoring** (also called "fixing" or scale linking) constrains selected parameter estimates to pre-specified values, placing the current analysis on a previously established scale. This is essential when comparing results across administrations, linking test forms, or monitoring rater drift over time.

This function applies the same preprocessing and key-resolution rules as `fit_mfrm()`, but returns an audit object so constraints can be checked *before* estimation. Running the audit first helps avoid estimation failures caused by misspecified or data-incompatible anchors.

**Anchor types:**

- *Direct anchors* fix individual element measures to specific logit values (e.g., Rater R1 anchored at 0.35 logits).
- *Group anchors* constrain the mean of a set of elements to a target value, allowing individual elements to vary freely around that mean.
- When both types overlap for the same element, the direct anchor takes precedence.

**Design checks** verify that each anchored element has at least `min_obs_per_element` weighted observations (default 30) and each score category has at least `min_obs_per_category` (default 10). These thresholds follow standard Rasch sample-size recommendations (Linacre, 1994).

**Value**

A list of class `mfrm_anchor_audit` with:

- `anchors`: cleaned anchor table used by estimation
- `group_anchors`: cleaned group-anchor table used by estimation
- `facet_summary`: counts of levels, constrained levels, and free levels
- `design_checks`: observation-count checks by level/category
- `thresholds`: active threshold settings used for recommendations
- `issue_counts`: issue-type counts
- `issues`: list of issue tables
- `recommendations`: package-native anchor guidance strings

**Interpreting output**

- `issue_counts/issues`: concrete data or specification problems.
- `facet_summary`: constraint coverage by facet.
- `design_checks`: whether anchor targets have enough observations.
- `recommendations`: action items before estimation.

**Typical workflow**

1. Build candidate anchors (e.g., with `make_anchor_table()`).
2. Run `audit_mfrm_anchors(...)`.
3. Resolve issues, then fit with `fit_mfrm()`.

**See Also**

`fit_mfrm()`, `describe_mfrm_data()`, `make_anchor_table()`

**Examples**

```
toy <- load_mfrm_data("example_core")

anchors <- data.frame(
  Facet = c("Rater", "Rater"),
  Level = c("R1", "R1"),
  Anchor = c(0, 0.1),
  stringsAsFactors = FALSE
)
aud <- audit_mfrm_anchors(
  data = toy,
  person = "Person",
  facets = c("Rater", "Criterion"),
  score = "Score",
  anchors = anchors
)
aud$issue_counts
```

```
summary(aud)
p_aud <- plot(aud, draw = FALSE)
class(p_aud)
```

---

bias_count_table	<i>Build a bias-cell count report</i>
------------------	---------------------------------------

---

## Description

Build a bias-cell count report

## Usage

```
bias_count_table(
  bias_results,
  min_count_warn = 10,
  branch = c("original", "facets"),
  fit = NULL
)
```

## Arguments

bias_results	Output from <code>estimate_bias()</code> .
min_count_warn	Minimum count threshold for flagging sparse bias cells.
branch	Output branch: "facets" keeps legacy manual-aligned naming, "original" returns compact QC-oriented names.
fit	Optional <code>fit_mfrm()</code> result used to attach run context metadata.

## Details

This helper summarizes how many observations contribute to each bias-cell estimate and flags sparse cells.

Branch behavior:

- "facets": keeps legacy manual-aligned column labels (Sq, Observed Count, Obs-Exp Average, Model S.E.) for side-by-side comparison with external workflows.
- "original": keeps compact field names (Count, BiasSize, SE) for custom QC workflows and scripting.

## Value

A named list with:

- table: cell-level counts with low-count flags
- by\_facet: named list of counts aggregated by each interaction facet
- by\_facet\_a, by\_facet\_b: first two facet summaries (legacy compatibility)

- `summary`: one-row summary
- `thresholds`: applied thresholds
- `branch, style`: output branch metadata
- `fit_overview`: optional one-row fit metadata when `fit` is supplied

### Interpreting output

- `table`: cell-level contribution counts and low-count flags.
- `by_facet`: sparse-cell structure by each interaction facet.
- `summary`: overall low-count prevalence.
- `fit_overview`: optional run context (when `fit` is supplied).

Low-count cells should be interpreted cautiously because bias-size estimates can become unstable with sparse support.

### Typical workflow

1. Estimate bias with `estimate_bias()`.
2. Build `bias_count_table(...)` in desired branch.
3. Review low-count flags before interpreting bias magnitudes.

### Further guidance

For a plot-selection guide and a longer walkthrough, see [mfrmr\\_visual\\_diagnostics](#) and `vignette("mfrmr-visual-diagnostics", package = "mfrmr")`.

### Output columns

The `table` data.frame contains, in the legacy-compatible branch:

**FacetA, FacetB** Interaction facet level identifiers; placeholder names for the two interaction facets.

**Sq** Sequential row number.

**Obsrvd Count** Number of observations for this cell.

**Obs-Exp Average** Observed minus expected average for this cell.

**Model S.E.** Standard error of the bias estimate.

**Infit, Outfit** Fit statistics for this cell.

**LowCountFlag** Logical; TRUE when `count < min_count_warn`.

The `summary` data.frame contains:

**InteractionFacets** Names of the interaction facets.

**Cells, TotalCount** Number of cells and total observations.

**LowCountCells, LowCountPercent** Number and share of low-count cells.

### See Also

[estimate\\_bias\(\)](#), [unexpected\\_after\\_bias\\_table\(\)](#), [build\\_fixed\\_reports\(\)](#), [mfrmr\\_visual\\_diagnostics](#)

**Examples**

```

toy <- load_mfrmr_data("example_bias")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
bias <- estimate_bias(fit, diag, facet_a = "Rater", facet_b = "Criterion", max_iter = 2)
t11 <- bias_count_table(bias)
t11_facets <- bias_count_table(bias, branch = "facets", fit = fit)
summary(t11)
p <- plot(t11, draw = FALSE)
p2 <- plot(t11, type = "lowcount_by_facet", draw = FALSE)
if (interactive()) {
  plot(
    t11,
    type = "cell_counts",
    draw = TRUE,
    main = "Bias Cell Counts (Customized)",
    palette = c(count = "#2b8cbe", low = "#cb181d"),
    label_angle = 45
  )
}

```

---

bias\_interaction\_report

*Build a bias-interaction plot-data bundle (preferred alias)*

---

**Description**

Build a bias-interaction plot-data bundle (preferred alias)

**Usage**

```

bias_interaction_report(
  x,
  diagnostics = NULL,
  facet_a = NULL,
  facet_b = NULL,
  interaction_facets = NULL,
  max_abs = 10,
  omit_extreme = TRUE,
  max_iter = 4,
  tol = 0.001,
  top_n = 50,
  abs_t_warn = 2,
  abs_bias_warn = 0.5,
  p_max = 0.05,
  sort_by = c("abs_t", "abs_bias", "prob")
)

```

**Arguments**

x	Output from <code>estimate_bias()</code> or <code>fit_mfrm()</code> .
diagnostics	Optional output from <code>diagnose_mfrm()</code> (used when x is fit).
facet_a	First facet name (required when x is fit and <code>interaction_facets</code> is not supplied).
facet_b	Second facet name (required when x is fit and <code>interaction_facets</code> is not supplied).
interaction_facets	Character vector of two or more facets.
max_abs	Bound for absolute bias size when estimating from fit.
omit_extreme	Omit extreme-only elements when estimating from fit.
max_iter	Iteration cap for bias estimation when x is fit.
tol	Convergence tolerance for bias estimation when x is fit.
top_n	Maximum number of ranked rows to keep.
abs_t_warn	Warning cutoff for absolute t statistics.
abs_bias_warn	Warning cutoff for absolute bias size.
p_max	Warning cutoff for p-values.
sort_by	Ranking key: "abs_t", "abs_bias", or "prob".

**Details**

Preferred bundle API for interaction-bias diagnostics. The function can:

- use a precomputed bias object from `estimate_bias()`, or
- estimate internally from `mfrm_fit` + facet specification.

**Value**

A named list with bias-interaction plotting/report components. Class: `mfrm_bias_interaction`.

**Interpreting output**

Focus on ranked rows where multiple screening criteria converge:

- large absolute t statistic
- large absolute bias size
- small screening tail area

The bundle is optimized for downstream `summary()` and `plot_bias_interaction()` views.

**Typical workflow**

1. Run `estimate_bias()` (or provide `mfrm_fit` here).
2. Build `bias_interaction_report(...)`.
3. Review `summary(out)` and visualize with `plot_bias_interaction()`.

**See Also**

[estimate\\_bias\(\)](#), [build\\_fixed\\_reports\(\)](#), [plot\\_bias\\_interaction\(\)](#)

**Examples**

```
toy <- load_mfrmr_data("example_bias")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
bias <- estimate_bias(fit, diag, facet_a = "Rater", facet_b = "Criterion", max_iter = 2)
out <- bias_interaction_report(bias, top_n = 10)
summary(out)
p_bi <- plot(out, draw = FALSE)
class(p_bi)
```

---

bias\_iteration\_report *Build a bias-iteration report*

---

**Description**

Build a bias-iteration report

**Usage**

```
bias_iteration_report(
  x,
  diagnostics = NULL,
  facet_a = NULL,
  facet_b = NULL,
  interaction_facets = NULL,
  max_abs = 10,
  omit_extreme = TRUE,
  max_iter = 4,
  tol = 0.001,
  top_n = 10
)
```

**Arguments**

x	Output from <a href="#">estimate_bias()</a> or <a href="#">fit_mfrm()</a> .
diagnostics	Optional output from <a href="#">diagnose_mfrm()</a> (used when x is fit).
facet_a	First facet name (required when x is fit and interaction_facets is not supplied).
facet_b	Second facet name (required when x is fit and interaction_facets is not supplied).
interaction_facets	Character vector of two or more facets.

max_abs	Bound for absolute bias size when estimating from fit.
omit_extreme	Omit extreme-only elements when estimating from fit.
max_iter	Iteration cap for bias estimation when x is fit.
tol	Convergence tolerance for bias estimation when x is fit.
top_n	Maximum number of iteration rows to keep in preview-oriented summaries. The full iteration table is always returned.

## Details

This report focuses on the recalibration path used by `estimate_bias()`. It provides a package-native counterpart to legacy iteration printouts by exposing the iteration table, convergence summary, and orientation audit in one bundle.

## Value

A named list with:

- `table`: iteration history
- `summary`: one-row convergence summary
- `orientation_audit`: interaction-facet sign audit
- `settings`: resolved reporting options

## See Also

[estimate\\_bias\(\)](#), [bias\\_interaction\\_report\(\)](#), [build\\_fixed\\_reports\(\)](#)

## Examples

```
toy <- load_mfrmr_data("example_bias")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
out <- bias_iteration_report(fit, diagnostics = diag, facet_a = "Rater", facet_b = "Criterion")
summary(out)
```

---

bias\_pairwise\_report *Build a bias pairwise-contrast report*

---

## Description

Build a bias pairwise-contrast report

**Usage**

```

bias_pairwise_report(
  x,
  diagnostics = NULL,
  facet_a = NULL,
  facet_b = NULL,
  interaction_facets = NULL,
  max_abs = 10,
  omit_extreme = TRUE,
  max_iter = 4,
  tol = 0.001,
  target_facet = NULL,
  context_facet = NULL,
  top_n = 50,
  p_max = 0.05,
  sort_by = c("abs_t", "abs_contrast", "prob")
)

```

**Arguments**

x	Output from <code>estimate_bias()</code> or <code>fit_mfrm()</code> .
diagnostics	Optional output from <code>diagnose_mfrm()</code> (used when x is fit).
facet_a	First facet name (required when x is fit and <code>interaction_facets</code> is not supplied).
facet_b	Second facet name (required when x is fit and <code>interaction_facets</code> is not supplied).
interaction_facets	Character vector of two or more facets.
max_abs	Bound for absolute bias size when estimating from fit.
omit_extreme	Omit extreme-only elements when estimating from fit.
max_iter	Iteration cap for bias estimation when x is fit.
tol	Convergence tolerance for bias estimation when x is fit.
target_facet	Facet whose local contrasts should be compared across the paired context facet. Defaults to the first interaction facet.
context_facet	Optional facet to condition on. Defaults to the other facet in a 2-way interaction.
top_n	Maximum number of ranked rows to keep.
p_max	Flagging cutoff for pairwise p-values.
sort_by	Ranking key: "abs_t", "abs_bias", or "prob".

**Details**

This helper exposes the pairwise contrast table that was previously only reachable through fixed-width output generation. It is available only for 2-way interactions. The pairwise contrast statistic uses a Welch/Satterthwaite approximation and is labeled as a Rasch-Welch comparison in the output metadata.

**Value**

A named list with:

- table: pairwise contrast rows
- summary: one-row contrast summary
- orientation\_audit: interaction-facet sign audit
- settings: resolved reporting options

**See Also**

[estimate\\_bias\(\)](#), [bias\\_interaction\\_report\(\)](#), [build\\_fixed\\_reports\(\)](#)

**Examples**

```
toy <- load_mfrmr_data("example_bias")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
out <- bias_pairwise_report(fit, diagnostics = diag, facet_a = "Rater", facet_b = "Criterion")
summary(out)
```

---

build\_apa\_outputs

*Build APA text outputs from model results*

---

**Description**

Build APA text outputs from model results

**Usage**

```
build_apa_outputs(
  fit,
  diagnostics,
  bias_results = NULL,
  context = list(),
  whexact = FALSE
)
```

**Arguments**

fit	Output from <a href="#">fit_mfrm()</a> .
diagnostics	Output from <a href="#">diagnose_mfrm()</a> .
bias_results	Optional output from <a href="#">estimate_bias()</a> .
context	Optional named list for report context.
whexact	Use exact ZSTD transformation.

## Details

context is an optional named list for narrative customization. Frequently used fields include:

- assessment, setting, scale\_desc
- rater\_training, raters\_per\_response
- rater\_facet (used for targeted reliability note text)
- line\_width (optional text wrapping width for report\_text; default = 92)

Output text includes residual-PCA screening commentary if PCA diagnostics are available in `diagnostics`.

By default, `report_text` includes:

- model/data design summary (N, facet counts, scale range)
- optimization/convergence metrics (Converged, Iterations, LogLik, AIC, BIC)
- anchor/constraint summary (noncenter\_facet, anchored levels, group anchors, dummy facets)
- category/threshold diagnostics (including disordered-step details when present)
- overall fit, misfit count, and top misfit levels
- facet reliability/separation, residual PCA summary, and bias-screen counts

## Value

An object of class `mfrm_apa_outputs` with:

- `report_text`: APA-style Method/Results draft prose
- `table_figure_notes`: consolidated draft notes for tables/visuals
- `table_figure_captions`: draft caption candidates without figure numbering
- `section_map`: package-native section table for manuscript assembly
- `contract`: structured APA reporting contract used for downstream checks

## Interpreting output

- `report_text`: manuscript-draft narrative covering Method (model specification, estimation, convergence) and Results (global fit, facet separation/reliability, misfit triage, category diagnostics, residual-PCA screening, bias screening). Written in third-person past tense following APA 7th edition conventions, but still intended for human review.
- `table_figure_notes`: reusable draft note blocks for table/figure appendices.
- `table_figure_captions`: draft caption candidates aligned to generated outputs.

When bias results or PCA diagnostics are not supplied, those sections are omitted from the narrative rather than producing placeholder text.

## Typical workflow

1. Build diagnostics (and optional bias results).
2. Run `build_apa_outputs(...)`.
3. Check `summary(apa)` for completeness.
4. Insert `apa$report_text` and note/caption fields into manuscript drafts after checking the listed cautions.

**Context template**

A minimal context list can include fields such as:

- `assessment`: name of the assessment task
- `setting`: administration context
- `scale_desc`: short description of the score scale
- `rater_facet`: rater facet label used in narrative reliability text

**See Also**

[build\\_visual\\_summaries\(\)](#), [estimate\\_bias\(\)](#), [reporting\\_checklist\(\)](#), [mfrmr\\_reporting\\_and\\_apa](#)

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "both")
apa <- build_apa_outputs(
  fit,
  diag,
  context = list(
    assessment = "Toy writing task",
    setting = "Demonstration dataset",
    scale_desc = "0-2 rating scale",
    rater_facet = "Rater"
  )
)
names(apa)
class(apa)
s_apa <- summary(apa)
s_apa$overview
chk <- reporting_checklist(fit, diagnostics = diag)
head(chk$checklist[, c("Section", "Item", "DraftReady", "NextAction")])
cat(apa$report_text)
apa$section_map[, c("SectionId", "Available")]
```

---

build\_equating\_chain *Build a screened linking chain across ordered calibrations*

---

**Description**

Links a series of calibration waves by computing mean offsets between adjacent pairs of fits. Common linking elements (e.g., raters or items that appear in consecutive administrations) are used to estimate the scale shift. Cumulative offsets place all waves on a common metric anchored to the first wave. The procedure is intended as a practical screened linking aid, not as a full general-purpose equating framework.

**Usage**

```

build_equating_chain(
  fits,
  anchor_facets = NULL,
  include_person = FALSE,
  drift_threshold = 0.5
)

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

## S3 method for class 'mfrm_equating_chain'
summary(object, ...)

## S3 method for class 'summary.mfrm_equating_chain'
print(x, ...)

```

**Arguments**

fits	Named list of mfrm_fit objects in chain order.
anchor_facets	Character vector of facets to use as linking elements.
include_person	Include person estimates in linking.
drift_threshold	Threshold for flagging large residuals in links.
x	An mfrm_equating_chain object.
...	Ignored.
object	An mfrm_equating_chain object (for summary).

**Details**

The screened linking chain uses a screened link-offset method. For each pair of adjacent waves  $(A, B)$ , the function:

1. Identifies common linking elements (facet levels present in both fits).
2. Computes per-element differences:

$$d_e = \hat{\delta}_{e,B} - \hat{\delta}_{e,A}$$

3. Computes a preliminary link offset using the inverse-variance weighted mean of these differences when standard errors are available (otherwise an unweighted mean).
4. Screens out elements whose residual from that preliminary offset exceeds drift\_threshold, then recomputes the final offset on the retained set.
5. Records Offset\_SD (standard deviation of retained residuals) and Max\_Residual (maximum absolute deviation from the mean) as indicators of link quality.
6. Flags links with fewer than 5 retained common elements in any linking facet as having thin support.

Cumulative offsets are computed by chaining link offsets from Wave 1 forward, placing all waves onto the metric of the first wave.

Elements whose per-link residual exceeds `drift_threshold` are flagged in `$element_detail$Flag`. A high `Offset_SD`, many flagged elements, or a thin retained anchor set signals an unstable link that may compromise the resulting scale placement.

### Value

Object of class `mfrm_equating_chain` with components:

**links** Tibble of link-level statistics (offset, SD, etc.).

**cumulative** Tibble of cumulative offsets per wave.

**element\_detail** Tibble of element-level linking details.

**common\_by\_facet** Tibble of retained common-element counts by facet.

**config** List of analysis configuration.

### Which function should I use?

- Use `anchor_to_baseline()` for a single new wave anchored to a known baseline.
- Use `detect_anchor_drift()` when you want direct comparison against one reference wave.
- Use `build_equating_chain()` when no single wave should dominate and you want ordered, adjacent links across the series.

### Interpreting output

- `$links`: one row per adjacent pair with `From`, `To`, `N_Common`, `N_Retained`, `Offset_Prelim`, `Offset`, `Offset_SD`, and `Max_Residual`. Small `Offset_SD` relative to the offset indicates a consistent shift across elements. `LinkSupportAdequate = FALSE` means at least one linking facet retained fewer than 5 common elements after screening.
- `$cumulative`: one row per wave with its cumulative offset from Wave 1. Wave 1 always has offset 0.
- `$element_detail`: per-element linking statistics (estimate in each wave, difference, residual from mean offset, and flag status). Flagged elements may indicate DIF or rater re-training effects.
- `$common_by_facet`: retained common-element counts by linking facet for each adjacent link.
- `$config`: records wave names and analysis parameters.
- Read links before cumulative: weak adjacent links can make later cumulative offsets less trustworthy.

### Typical workflow

1. Fit each administration wave separately: `fit_a <- fit_mfrm(...)`.
2. Combine into an ordered named list: `fits <- list(Spring23 = fit_s, Fall23 = fit_f, Spring24 = fit_s2)`.
3. Call `chain <- build_equating_chain(fits)`.
4. Review `summary(chain)` for link quality.

5. Visualize with `plot_anchor_drift(chain, type = "chain")`.
6. For problematic links, investigate flagged elements in `chain$element_detail` and consider removing them from the anchor set.

### See Also

`detect_anchor_drift()`, `anchor_to_baseline()`, `make_anchor_table()`, `plot_anchor_drift()`

### Examples

```
toy <- load_mfrmr_data("example_core")
people <- unique(toy$Person)
d1 <- toy[toy$Person %in% people[1:12], , drop = FALSE]
d2 <- toy[toy$Person %in% people[13:24], , drop = FALSE]
fit1 <- fit_mfrm(d1, "Person", c("Rater", "Criterion"), "Score",
  method = "JML", maxit = 10)
fit2 <- fit_mfrm(d2, "Person", c("Rater", "Criterion"), "Score",
  method = "JML", maxit = 10)
chain <- build_equating_chain(list(Form1 = fit1, Form2 = fit2))
summary(chain)
chain$cumulative
```

---

`build_fixed_reports`    *Build legacy-compatible fixed-width text reports*

---

### Description

Build legacy-compatible fixed-width text reports

### Usage

```
build_fixed_reports(
  bias_results,
  target_facet = NULL,
  branch = c("facets", "original")
)
```

### Arguments

<code>bias_results</code>	Output from <code>estimate_bias()</code> .
<code>target_facet</code>	Optional target facet for pairwise contrast table.
<code>branch</code>	Output branch: "facets" keeps the legacy-compatible fixed-width layout; "original" returns compact sectioned fixed-width text for internal reporting.

## Details

This function generates plain-text, fixed-width output intended to be read in console/log environments or exported into text reports.

The pairwise section (Table 14 style) is only generated for 2-way bias runs. For higher-order interactions (`interaction_facets` length  $\geq 3$ ), the function returns the bias table text and a note explaining why pairwise contrasts were skipped.

## Value

A named list with:

- `bias_fixed`: fixed-width interaction table text
- `pairwise_fixed`: fixed-width pairwise contrast text
- `pairwise_table`: underlying pairwise data.frame

## Interpreting output

- `bias_fixed`: fixed-width table of interaction effects.
- `pairwise_fixed`: pairwise contrast text (2-way only).
- `pairwise_table`: machine-readable contrast table.
- `interaction_label`: facets used for the bias run.

## Typical workflow

1. Run `estimate_bias()`.
2. Build text bundle with `build_fixed_reports(...)`.
3. Use `summary()/plot()` for quick checks, then export text blocks.

## Preferred route for new analyses

For new reporting workflows, prefer `bias_interaction_report()` and `build_apa_outputs()`. Use `build_fixed_reports()` when a fixed-width text artifact is specifically required for a compatibility handoff.

## See Also

[estimate\\_bias\(\)](#), [build\\_apa\\_outputs\(\)](#), [bias\\_interaction\\_report\(\)](#), [mfrmr\\_reports\\_and\\_tables](#), [mfrmr\\_compatibility\\_layer](#)

## Examples

```
toy <- load_mfrmr_data("example_bias")
fit <- fit_mfrmr(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrmr(fit, residual_pca = "none")
bias <- estimate_bias(fit, diag, facet_a = "Rater", facet_b = "Criterion", max_iter = 2)
fixed <- build_fixed_reports(bias)
fixed_original <- build_fixed_reports(bias, branch = "original")
summary(fixed)
```

```

p <- plot(fixed, draw = FALSE)
p2 <- plot(fixed, type = "pvalue", draw = FALSE)
if (interactive()) {
  plot(
    fixed,
    type = "contrast",
    draw = TRUE,
    main = "Pairwise Contrasts (Customized)",
    palette = c(pos = "#1b9e77", neg = "#d95f02"),
    label_angle = 45
  )
}

```

---

build\_mfrm\_manifest    *Build a reproducibility manifest for an MFRM analysis*

---

## Description

Build a reproducibility manifest for an MFRM analysis

## Usage

```

build_mfrm_manifest(
  fit,
  diagnostics = NULL,
  bias_results = NULL,
  population_prediction = NULL,
  unit_prediction = NULL,
  plausible_values = NULL,
  include_person_anchors = FALSE
)

```

## Arguments

fit	Output from <code>fit_mfrm()</code> or <code>run_mfrm_facets()</code> .
diagnostics	Optional output from <code>diagnose_mfrm()</code> . When NULL, diagnostics are computed with <code>residual_pca = "none"</code> .
bias_results	Optional output from <code>estimate_bias()</code> or a named list of bias bundles.
population_prediction	Optional output from <code>predict_mfrm_population()</code> .
unit_prediction	Optional output from <code>predict_mfrm_units()</code> .
plausible_values	Optional output from <code>sample_mfrm_plausible_values()</code> .
include_person_anchors	If TRUE, include person measures in the exported anchor table.

## Details

This helper captures the package-native equivalent of the Streamlit app's configuration export. It summarizes analysis settings, source columns, anchoring information, and which downstream outputs are currently available.

## Value

A named list with class `mfrm_manifest`.

## When to use this

Use `build_mfrm_manifest()` when you want a compact, machine-readable record of how an analysis was run. Compared with related helpers:

- `export_mfrm()` writes analysis tables only.
- `build_mfrm_manifest()` records settings and available outputs.
- `build_mfrm_replay_script()` creates an executable R script.
- `export_mfrm_bundle()` writes a shareable folder of files.

## Output

The returned bundle has class `mfrm_manifest` and includes:

- `summary`: one-row analysis overview
- `environment`: package/R/platform metadata
- `model_settings`: key-value model settings table
- `source_columns`: key-value data-column table
- `estimation_control`: key-value optimizer settings table
- `anchor_summary`: facet-level anchor summary
- `anchors`: machine-readable anchor table
- `available_outputs`: availability table for diagnostics/bias/PCA/prediction outputs
- `settings`: manifest build settings

## Interpreting output

The summary table is the quickest place to confirm that you are looking at the intended analysis. The `model_settings`, `source_columns`, and `estimation_control` tables are designed for audit trails and method write-up. The `available_outputs` table is especially useful before building bundles, because it tells you whether residual PCA, anchors, bias results, or prediction-side artifacts are already available. A practical reading order is `summary` first, `available_outputs` second, and `anchors` last when reproducibility depends on fixed constraints.

**Typical workflow**

1. Fit a model with `fit_mfrm()` or `run_mfrm_facets()`.
2. Compute diagnostics once with `diagnose_mfrm()` if you want explicit control over residual PCA.
3. Build a manifest and inspect summary plus `available_outputs`.
4. If you need files on disk, pass the same objects to `export_mfrm_bundle()`.

**See Also**

`export_mfrm_bundle()`, `build_mfrm_replay_script()`, `make_anchor_table()`, `reporting_checklist()`

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score",
               method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
manifest <- build_mfrm_manifest(fit, diagnostics = diag)
manifest$summary[, c("Model", "Method", "Observations", "Facets")]
manifest$available_outputs[, c("Component", "Available")]
```

---

`build_mfrm_replay_script`

*Build a package-native replay script for an MFRM analysis*

---

**Description**

Build a package-native replay script for an MFRM analysis

**Usage**

```
build_mfrm_replay_script(
  fit,
  diagnostics = NULL,
  bias_results = NULL,
  population_prediction = NULL,
  unit_prediction = NULL,
  plausible_values = NULL,
  data_file = "your_data.csv",
  script_mode = c("auto", "fit", "facets"),
  include_bundle = FALSE,
  bundle_dir = "analysis_bundle",
  bundle_prefix = "mfrmr_replay"
)
```

**Arguments**

fit	Output from <code>fit_mfrm()</code> or <code>run_mfrm_facets()</code> .
diagnostics	Optional output from <code>diagnose_mfrm()</code> . When NULL, diagnostics are reused from <code>run_mfrm_facets()</code> when available, otherwise recomputed.
bias_results	Optional output from <code>estimate_bias()</code> or a named list of bias bundles. When supplied, the generated script includes package-native bias estimation calls.
population_prediction	Optional output from <code>predict_mfrm_population()</code> to recreate in the generated script.
unit_prediction	Optional output from <code>predict_mfrm_units()</code> to recreate in the generated script.
plausible_values	Optional output from <code>sample_mfrm_plausible_values()</code> to recreate in the generated script.
data_file	Path to the analysis data file used in the generated script.
script_mode	One of "auto", "fit", or "facets". "auto" uses <code>run_mfrm_facets()</code> when the input object came from that workflow.
include_bundle	If TRUE, append an <code>export_mfrm_bundle()</code> call to the generated script.
bundle_dir	Output directory used when <code>include_bundle = TRUE</code> .
bundle_prefix	Prefix used by the generated bundle exporter call.

**Details**

This helper mirrors the Streamlit app's reproducible-download idea, but uses `mfrm`'s installed API rather than embedding a separate estimation engine. The generated script assumes the user has the package installed and provides a data file at `data_file`.

Anchor and group-anchor constraints are embedded directly from the fitted object's stored configuration, so the script can replay anchored analyses without manual table reconstruction.

**Value**

A named list with class `mfrm_replay_script`.

**When to use this**

Use `build_mfrm_replay_script()` when you want a package-native recipe that another analyst can rerun later. Compared with related helpers:

- `build_mfrm_manifest()` records settings but does not run anything.
- `build_mfrm_replay_script()` produces executable R code.
- `export_mfrm_bundle()` can optionally write the replay script to disk.

## Interpreting output

The returned object contains:

- `summary`: a one-row overview of the chosen replay mode and whether bundle export was included
- `script`: the generated R code as a single string
- `anchors` and `group_anchors`: the exact stored constraints that were embedded into the script

If `ScriptMode` is "facets", the script replays the higher-level `run_mfrm_facets()` workflow. If it is "fit", the script uses `fit_mfrm()` directly.

## Mode guide

- "auto" is the safest default and follows the structure of the supplied object.
- "fit" is useful when you want a minimal script centered on `fit_mfrm()`.
- "facets" is useful when you want to preserve the higher-level `run_mfrm_facets()` workflow, including stored column mapping.

## Typical workflow

1. Finalize a fit and diagnostics object.
2. Generate the replay script with the path you want users to read from.
3. Write `replay$script` to disk, or let `export_mfrm_bundle()` do it for you.
4. Rerun the script in a fresh R session to confirm reproducibility.

## See Also

`build_mfrm_manifest()`, `export_mfrm_bundle()`, `run_mfrm_facets()`

## Examples

```
toy <- load_mfrm_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score",
               method = "JML", maxit = 25)
replay <- build_mfrm_replay_script(fit, data_file = "your_data.csv")
replay$summary[, c("ScriptMode", "ResidualPCA", "BiasPairs")]
cat(substr(replay$script, 1, 120))
```

---

build\_mfrm\_sim\_spec     *Build an explicit simulation specification for MFRM design studies*

---

### Description

Build an explicit simulation specification for MFRM design studies

### Usage

```
build_mfrm_sim_spec(
  n_person = 50,
  n_rater = 4,
  n_criterion = 4,
  raters_per_person = n_rater,
  score_levels = 4,
  theta_sd = 1,
  rater_sd = 0.35,
  criterion_sd = 0.25,
  noise_sd = 0,
  step_span = 1.4,
  thresholds = NULL,
  model = c("RSM", "PCM"),
  step_facet = "Criterion",
  assignment = c("crossed", "rotating", "resampled", "skeleton"),
  latent_distribution = c("normal", "empirical"),
  empirical_person = NULL,
  empirical_rater = NULL,
  empirical_criterion = NULL,
  assignment_profiles = NULL,
  design_skeleton = NULL,
  group_levels = NULL,
  dif_effects = NULL,
  interaction_effects = NULL
)
```

### Arguments

n_person	Number of persons/respondents to generate.
n_rater	Number of rater facet levels to generate.
n_criterion	Number of criterion/item facet levels to generate.
raters_per_person	Number of raters assigned to each person.
score_levels	Number of ordered score categories.
theta_sd	Standard deviation of simulated person measures.
rater_sd	Standard deviation of simulated rater severities.

criterion_sd	Standard deviation of simulated criterion difficulties.
noise_sd	Optional observation-level noise added to the linear predictor.
step_span	Spread used to generate equally spaced thresholds when thresholds = NULL.
thresholds	Optional threshold specification. Use either a numeric vector of common thresholds or a data frame with columns StepFacet, Step/StepIndex, and Estimate.
model	Measurement model recorded in the simulation specification.
step_facet	Step facet used when model = "PCM" and threshold values vary across levels.
assignment	Assignment design. "crossed" means every person sees every rater; "rotating" uses a balanced rotating subset; "resampled" reuses empirical person-level rater-assignment profiles; "skeleton" reuses an observed person-by-facet design skeleton.
latent_distribution	Latent-value generator. "normal" samples from centered normal distributions using the supplied standard deviations. "empirical" resamples centered support values from empirical_person/empirical_rater/empirical_criterion.
empirical_person	Optional numeric support values used when latent_distribution = "empirical".
empirical_rater	Optional numeric support values used when latent_distribution = "empirical".
empirical_criterion	Optional numeric support values used when latent_distribution = "empirical".
assignment_profiles	Optional data frame with columns TemplatePerson and Rater (optionally Group) describing empirical person-level rater-assignment profiles used when assignment = "resampled".
design_skeleton	Optional data frame with columns TemplatePerson, Rater, and Criterion (optionally Group and Weight) describing an observed response skeleton used when assignment = "skeleton".
group_levels	Optional character vector of group labels.
dif_effects	Optional data frame of true group-linked DIF effects.
interaction_effects	Optional data frame of true interaction effects.

## Details

build\_mfrm\_sim\_spec() creates an explicit, portable simulation specification that can be passed to simulate\_mfrm\_data(). The goal is to make the data-generating mechanism inspectable and reusable rather than relying only on ad hoc scalar arguments.

The resulting object records:

- design counts (n\_person, n\_rater, n\_criterion, raters\_per\_person)
- latent spread assumptions (theta\_sd, rater\_sd, criterion\_sd)
- optional empirical latent support values for semi-parametric simulation

- threshold structure (threshold\_table)
- assignment design (assignment)
- optional empirical assignment profiles (assignment\_profiles) with optional person-level Group labels
- optional observed response skeleton (design\_skeleton) with optional person-level Group labels and observation-level Weight values
- optional signal tables for DIF and interaction bias

The current generator still targets the package's standard person x rater x criterion workflow. When threshold values are provided by StepFacet, the supported step facets are the generated Rater or Criterion levels.

### Value

An object of class `mfrm_sim_spec`.

### Interpreting output

This object does not contain simulated data. It is a data-generating specification that tells `simulate_mfrm_data()` how to generate them.

### See Also

`extract_mfrm_sim_spec()`, `simulate_mfrm_data()`

### Examples

```
spec <- build_mfrm_sim_spec(  
  n_person = 30,  
  n_rater = 4,  
  n_criterion = 3,  
  raters_per_person = 2,  
  assignment = "rotating"  
)  
spec$model  
spec$assignment  
spec$threshold_table
```

---

`build_visual_summaries`

*Build warning and narrative summaries for visual outputs*

---

### Description

Build warning and narrative summaries for visual outputs

**Usage**

```
build_visual_summaries(
  fit,
  diagnostics,
  threshold_profile = "standard",
  thresholds = NULL,
  summary_options = NULL,
  whexact = FALSE,
  branch = c("original", "facets")
)
```

**Arguments**

fit	Output from <code>fit_mfrm()</code> .
diagnostics	Output from <code>diagnose_mfrm()</code> .
threshold_profile	Threshold profile name (strict, standard, lenient).
thresholds	Optional named overrides for profile thresholds.
summary_options	Summary options for <code>build_visual_summary_map()</code> .
whexact	Use exact ZSTD transformation.
branch	Output branch: "facets" adds FACETS crosswalk metadata for manual-aligned reporting; "original" keeps package-native summary output.

**Details**

This function returns visual-keyed text maps to support dashboard/report rendering without hard-coding narrative strings in UI code.

thresholds can override any profile field by name. Common overrides:

- n\_obs\_min, n\_person\_min
- misfit\_ratio\_warn, zstd2\_ratio\_warn, zstd3\_ratio\_warn
- pca\_first\_eigen\_warn, pca\_first\_prop\_warn

summary\_options supports:

- detail: "standard" or "detailed"
- max\_facet\_ranges: max facet-range snippets shown in visual summaries
- top\_misfit\_n: number of top misfit entries included

**Value**

An object of class `mfrm_visual_summaries` with:

- warning\_map: visual-level warning text vectors
- summary\_map: visual-level descriptive text vectors

- warning\_counts, summary\_counts: message counts by visual key
- crosswalk: FACETS-reference mapping for main visual keys
- branch, style, threshold\_profile: branch metadata

### Interpreting output

- warning\_map: rule-triggered warning text by visual key.
- summary\_map: descriptive narrative text by visual key.
- warning\_counts / summary\_counts: message-count tables for QA checks.

### Typical workflow

1. inspect defaults with `mfrm_threshold_profiles()`
2. choose `threshold_profile` (strict / standard / lenient)
3. optionally override selected fields via `thresholds`
4. pass result maps to report/dashboard rendering logic

### See Also

`mfrm_threshold_profiles()`, `build_apas_outputs()`

### Examples

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "both")
vis <- build_visual_summaries(fit, diag, threshold_profile = "strict")
vis2 <- build_visual_summaries(
  fit,
  diag,
  threshold_profile = "standard",
  thresholds = c(misfit_ratio_warn = 0.20, pca_first_eigen_warn = 2.0),
  summary_options = list(detail = "detailed", top_misfit_n = 5)
)
vis_facets <- build_visual_summaries(fit, diag, branch = "facets")
vis_facets$branch
summary(vis)
p <- plot(vis, type = "comparison", draw = FALSE)
p2 <- plot(vis, type = "warning_counts", draw = FALSE)
if (interactive()) {
  plot(
    vis,
    type = "comparison",
    draw = TRUE,
    main = "Warning vs Summary Counts (Customized)",
    palette = c(warning = "#cb181d", summary = "#3182bd"),
    label_angle = 45
  )
}
```

---

`category_curves_report`*Build a category curve export bundle (preferred alias)*

---

## Description

Build a category curve export bundle (preferred alias)

## Usage

```
category_curves_report(  
  fit,  
  theta_range = c(-6, 6),  
  theta_points = 241,  
  digits = 4,  
  include_fixed = FALSE,  
  fixed_max_rows = 400  
)
```

## Arguments

<code>fit</code>	Output from <code>fit_mfrm()</code> .
<code>theta_range</code>	Theta/logit range for curve coordinates.
<code>theta_points</code>	Number of points on the theta grid.
<code>digits</code>	Rounding digits for numeric graph output.
<code>include_fixed</code>	If TRUE, include a legacy-compatible fixed-width text block.
<code>fixed_max_rows</code>	Maximum rows shown in fixed-width graph tables.

## Details

Preferred high-level API for category-probability curve exports. Returns tidy curve coordinates and summary metadata for quick plotting/report integration without calling low-level helpers directly.

## Value

A named list with category-curve components. Class: `mfrm_category_curves`.

## Interpreting output

Use this report to inspect:

- where each category has highest probability across theta
- whether adjacent categories cross in expected order
- whether probability bands look compressed (often sparse categories)

Recommended read order:

1. `summary(out)` for compact diagnostics.
2. `out$curve_points` (or equivalent curve table) for downstream graphics.
3. `plot(out)` for a default visual check.

### Typical workflow

1. Fit model with `fit_mfrm()`.
2. Run `category_curves_report()` with suitable `theta_points`.
3. Use `summary()` and `plot()`; export tables for manuscripts/dashboard use.

### See Also

[category\\_structure\\_report\(\)](#), [rating\\_scale\\_table\(\)](#), [plot.mfrm\\_fit\(\)](#), [mfrm\\_reports\\_and\\_tables](#), [mfrm\\_visual\\_diagnostics](#)

### Examples

```
toy <- load_mfrm_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
out <- category_curves_report(fit, theta_points = 101)
summary(out)
names(out)
p_cc <- plot(out, draw = FALSE)
class(p_cc)
```

---

`category_structure_report`

*Build a category structure report (preferred alias)*

---

### Description

Build a category structure report (preferred alias)

### Usage

```
category_structure_report(
  fit,
  diagnostics = NULL,
  theta_range = c(-6, 6),
  theta_points = 241,
  drop_unused = FALSE,
  include_fixed = FALSE,
  fixed_max_rows = 200
)
```

**Arguments**

<code>fit</code>	Output from <code>fit_mfrm()</code> .
<code>diagnostics</code>	Optional output from <code>diagnose_mfrm()</code> .
<code>theta_range</code>	Theta/logit range used to derive transition points.
<code>theta_points</code>	Number of grid points used for transition-point search.
<code>drop_unused</code>	If TRUE, remove zero-count categories from outputs.
<code>include_fixed</code>	If TRUE, include a legacy-compatible fixed-width text block.
<code>fixed_max_rows</code>	Maximum rows per fixed-width section.

**Details**

Preferred high-level API for category-structure diagnostics. This wraps the legacy-compatible bar/transition export and returns a stable bundle interface for reporting and plotting.

**Value**

A named list with category-structure components. Class: `mfrm_category_structure`.

**Interpreting output**

Key components include:

- category usage/fit table (count, expected, infit/outfit, ZSTD)
- threshold ordering and adjacent threshold gaps
- category transition-point table on the requested theta grid

Practical read order:

1. `summary(out)` for compact warnings and threshold ordering.
2. `out$category_table` for sparse/misfitting categories.
3. `out$transition_points` to inspect crossing structure.
4. `plot(out)` for quick visual check.

**Typical workflow**

1. `fit_mfrm()` -> model.
2. `diagnose_mfrm()` -> residual/fit diagnostics (optional argument here).
3. `category_structure_report()` -> category health snapshot.
4. `summary()` and `plot()` for draft-oriented review of category structure.

**See Also**

[rating\\_scale\\_table\(\)](#), [category\\_curves\\_report\(\)](#), [plot.mfrm\\_fit\(\)](#), [mfrm\\_reports\\_and\\_tables](#), [mfrm\\_visual\\_diagnostics](#)

**Examples**

```

toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
out <- category_structure_report(fit)
summary(out)
names(out)
p_cs <- plot(out, draw = FALSE)
class(p_cs)

```

---

compare\_mfrm

*Compare two or more fitted MFRM models*


---

**Description**

Produce a side-by-side comparison of multiple `fit_mfrm()` results using information criteria, log-likelihood, and parameter counts. When exactly two nested models are supplied, a likelihood-ratio test is included.

**Usage**

```
compare_mfrm(..., labels = NULL, warn_constraints = TRUE, nested = FALSE)
```

**Arguments**

<code>...</code>	Two or more <code>mfrm_fit</code> objects to compare.
<code>labels</code>	Optional character vector of labels for each model. If <code>NULL</code> , labels are generated from model/method combinations.
<code>warn_constraints</code>	Logical. If <code>TRUE</code> (the default), emit a warning when models use different centering constraints ( <code>noncenter_facet</code> or <code>dummy_facets</code> ), which can make information-criterion comparisons misleading.
<code>nested</code>	Logical. Set to <code>TRUE</code> only when the supplied models are known to be nested and fitted with the same likelihood basis on the same observations. The default is <code>FALSE</code> , in which case no likelihood-ratio test is reported. When <code>TRUE</code> , the function still runs a conservative structural audit and computes the LRT only for supported nesting patterns.

**Details**

Models should be fit to the **same data** (same rows, same person/facet columns) for the comparison to be meaningful. The function checks that observation counts match and warns otherwise.

Information-criterion ranking is reported only when all candidate models use the package's MML estimation path, analyze the same observations, and converge successfully. Raw AIC and BIC values are still shown for each model, but `Delta_*`, weights, and preferred-model summaries are suppressed when the likelihood basis is not comparable enough for primary reporting.

**Nesting:** Two models are *nested* when one is a special case of the other obtained by imposing equality constraints. The most common nesting in MFRM is RSM (shared thresholds) inside PCM (item-specific thresholds). Models that differ only in estimation method (MML vs JML) on the same specification are not nested in the usual sense—use information criteria rather than LRT for that comparison.

The **likelihood-ratio test (LRT)** is reported only when exactly two models are supplied, `nested = TRUE`, the structural audit passes, and the difference in the number of parameters is positive:

$$\Lambda = -2(\ell_{\text{restricted}} - \ell_{\text{full}}) \sim \chi_{\Delta p}^2$$

The LRT is asymptotically valid when models are nested and the data are independent. With small samples or boundary conditions (e.g., variance components near zero), treat p-values as approximate.

## Value

An object of class `mfrm_comparison` (named list) with:

- `table`: data.frame of model-level statistics (LogLik, AIC, BIC, Delta\_AIC, AkaikeWeight, Delta\_BIC, BICWeight, npar, nobs, Model, Method, Converged, ICComparable).
- `lrt`: data.frame with likelihood-ratio test result (only when two models are supplied and `nested = TRUE`). Contains ChiSq, df, p\_value.
- `evidence_ratios`: data.frame of pairwise Akaike-weight ratios (Model1, Model2, EvidenceRatio). NULL when weights cannot be computed.
- `preferred`: named list with the preferred model label by each criterion.
- `comparison_basis`: list describing whether IC and LRT comparisons were considered comparable. Includes a conservative `nesting_audit`.

## Information-criterion diagnostics

In addition to raw AIC and BIC values, the function computes:

- **Delta\_AIC / Delta\_BIC**: difference from the best (minimum) value. A Delta < 2 is typically considered negligible; 4–7 suggests moderate evidence; > 10 indicates strong evidence against the higher-scoring model (Burnham & Anderson, 2002).
- **AkaikeWeight / BICWeight**: model probabilities derived from  $\exp(-0.5 * \text{Delta})$ , normalised across the candidate set. An Akaike weight of 0.90 means the model has a 90% being the best in the candidate set.
- **Evidence ratios**: pairwise ratios of Akaike weights, quantifying the relative evidence for one model over another (e.g., an evidence ratio of 5 means the preferred model is 5 times more likely).

AIC penalises complexity less than BIC; when they disagree, AIC favours the more complex model and BIC the simpler one.

**What this comparison means**

compare\_mfrm() is a same-basis model-comparison helper. Its strongest claims apply only when the models were fit to the same response data, under a compatible likelihood basis, and with compatible constraint structure.

**What this comparison does not justify**

- Do not treat AIC/BIC differences as primary evidence when table\$ICComparable is FALSE.
- Do not interpret the LRT unless nested = TRUE and the structural audit in comparison\_basis\$nesting\_audit passes.
- Do not compare models fit to different datasets, different score codings, or materially different constraint systems as if they were commensurate.

**Interpreting output**

- Lower AIC/BIC values indicate better parsimony-accuracy trade-off only when table\$ICComparable is TRUE.
- A significant LRT p-value suggests the more complex model provides a meaningfully better fit only when the nesting assumption truly holds.
- preferred indicates the model preferred by each criterion.
- evidence\_ratios gives pairwise Akaike-weight ratios (returned only when Akaike weights can be computed for at least two models).
- When comparing more than two models, interpret evidence ratios cautiously—they do not adjust for multiple comparisons.

**How to read the main outputs**

- table: first-pass comparison table; start with ICComparable, Model, Method, AIC, and BIC.
- comparison\_basis: records whether IC and LRT claims are defensible for the supplied models.
- lrt: nested-model test summary, present only when the requested and audited conditions are met.
- preferred: candidate preferred by each criterion when those summaries are available.

**Recommended next step**

Inspect comparison\_basis before writing conclusions. If comparability is weak, treat the result as descriptive and revise the model setup (for example, explicit step\_facet, common data, or common constraints) before using IC or LRT results in reporting.

**Typical workflow**

1. Fit two models with fit\_mfrm() (e.g., RSM and PCM).
2. Compare with compare\_mfrm(fit\_rsm, fit\_pcm).
3. Inspect summary(comparison) for AIC/BIC diagnostics and, when appropriate, an LRT.

**See Also**

[fit\\_mfrm\(\)](#), [diagnose\\_mfrm\(\)](#)

**Examples**

```
toy <- load_mfrmr_data("example_core")

fit_rsm <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score",
  method = "MML", model = "RSM", maxit = 25)
fit_pcm <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score",
  method = "MML", model = "PCM",
  step_facet = "Criterion", maxit = 25)
comp <- compare_mfrm(fit_rsm, fit_pcm, labels = c("RSM", "PCM"))
comp$table
comp$evidence_ratios
```

---

compute\_information     *Compute design-weighted precision curves for RSM fits*

---

**Description**

Calculates design-weighted score-variance curves across the latent trait (theta) for a fitted RSM many-facet Rasch model. Returns both an overall precision curve (`$tif`) and per-facet-level contribution curves (`$iif`) based on the realized observation pattern.

**Usage**

```
compute_information(fit, theta_range = c(-6, 6), theta_points = 201L)
```

**Arguments**

<code>fit</code>	Output from <a href="#">fit_mfrm()</a> .
<code>theta_range</code>	Numeric vector of length 2 giving the range of theta values. Default <code>c(-6, 6)</code> .
<code>theta_points</code>	Integer number of points at which to evaluate information. Default 201.

**Details**

For a polytomous Rasch model with  $K+1$  categories, the score variance at theta for one observed design cell is:

$$I(\theta) = \sum_{k=0}^K P_k(\theta) (k - E(\theta))^2$$

where  $P_k$  is the category probability and  $E(\theta)$  is the expected score at theta. In `mfrmr`, these cell-level variances are then aggregated with weights taken from the realized observation counts in `fit$prep$data`.

The resulting total curve is therefore a design-weighted precision screen rather than a pure textbook test-information function for an abstract fixed item set. The associated standard error summary is still  $SE(\theta) = 1/\sqrt{I(\theta)}$  for positive information values.

**Value**

An object of class `mfrm_information` (named list) with:

- `tif`: tibble with columns `Theta`, `Information`, `SE`. The `Information` column stores the design-weighted precision value.
- `iif`: tibble with columns `Theta`, `Facet`, `Level`, `Information`, and `Exposure`. Here too, `Information` stores a design-weighted contribution value retained under that column name for compatibility.
- `theta_range`: the evaluated theta range.

**What `tif` and `iif` mean here**

In `mfrm`, this helper currently supports only RSM fits. The total curve (`$tif`) is the sum of design-weighted cell contributions across all non-person facet levels in the fitted model. The facet-level contribution curves (`$iif`) keep those weighted contributions separated, so you can see which observed rater levels, criteria, or other facet levels are driving precision at different parts of the scale.

**What this quantity does not justify**

- It is not a textbook many-facet test-information function for an abstract fixed item set.
- It should not be used as if it were design-free evidence about a form's precision independent of the realized observation pattern.
- It does not currently extend to PCM fits; the helper stops for `model = "PCM"`.

**When to use this**

Use `compute_information()` when you want a design-weighted precision screen for an RSM fit along the latent continuum. In practice:

- start with the total precision curve for overall targeting across the realized observation pattern
- inspect facet-level contribution curves when you want to see which raters, criteria, or other facet levels account for more of that design-weighted precision
- widen `theta_range` if you expect extreme measures and want to inspect the tails explicitly

**Choosing the theta grid**

The defaults (`theta_range = c(-6, 6)`, `theta_points = 201`) work well for routine inspection. Expand the range if person or facet measures extend into the tails, and increase `theta_points` only when you need a smoother grid for reporting or custom graphics.

**Interpreting output**

- `$tif`: design-weighted precision curve data with `theta`, `Information`, and `SE`.
- `$iif`: design-weighted facet-level contribution curves for an RSM fit.
- Higher information implies more precise measurement at that theta.
- `SE` is inversely related to information.
- Peaks in the total curve show the trait region where the realized calibration is most informative.
- Facet-level curves help explain *which observed facet levels* contribute to those peaks; they are not standalone item-information curves and should be read as design contributions.

### How to read the main columns

- Theta: point on the latent continuum where the curve is evaluated.
- Information: design-weighted precision value at that theta.
- SE: approximate  $1 / \sqrt{\text{Information}}$  summary for positive values.
- Exposure: total realized observation weight contributing to a facet-level curve in `$iif`.

### Recommended next step

Compare the precision peak with person/facet locations from a Wright map or related diagnostics. If you need to decide how strongly SE/CI language can be used in reporting, follow with `precision_audit_report()`.

### Typical workflow

1. Fit a model with `fit_mfrm()`.
2. Run `compute_information(fit)`.
3. Plot with `plot_information(info, type = "tif")`.
4. If needed, inspect facet contributions with `plot_information(info, type = "iif", facet = "Rater")`.

### See Also

`fit_mfrm()`, `plot_information()`

### Examples

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score",
               method = "JML", model = "RSM", maxit = 25)
info <- compute_information(fit)
head(info$tif)
info$tif$Theta[which.max(info$tif$Information)]
```

---

data\_quality\_report    *Build a data quality summary report (preferred alias)*

---

### Description

Build a data quality summary report (preferred alias)

## Usage

```
data_quality_report(  
  fit,  
  data = NULL,  
  person = NULL,  
  facets = NULL,  
  score = NULL,  
  weight = NULL,  
  include_fixed = FALSE  
)
```

## Arguments

fit	Output from <a href="#">fit_mfrm()</a> .
data	Optional raw data frame used for row-level audit.
person	Optional person column name in data.
facets	Optional facet column names in data.
score	Optional score column name in data.
weight	Optional weight column name in data.
include_fixed	If TRUE, include a legacy-compatible fixed-width text block.

## Details

`summary(out)` is supported through `summary()`. `plot(out)` is dispatched through `plot()` for class `mfrm_data_quality` (`type = "row_audit", "category_counts", "missing_rows"`).

## Value

A named list with data-quality report components. Class: `mfrm_data_quality`.

## Interpreting output

- `summary`: retained/dropped row overview.
- `row_audit`: reason-level breakdown for data issues.
- `category_counts`: post-filter category usage.
- `unknown_elements`: facet levels in raw data but not in fitted design.

## Typical workflow

1. Run `data_quality_report(...)` with raw data.
2. Check row-audit and missing/unknown element sections.
3. Resolve issues before final estimation/reporting.

## See Also

[fit\\_mfrm\(\)](#), [describe\\_mfrm\\_data\(\)](#), [specifications\\_report\(\)](#), [mfrm\\_reports\\_and\\_tables](#), [mfrm\\_compatibility\\_layer](#)

**Examples**

```

toy <- load_mfrm_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
out <- data_quality_report(
  fit, data = toy, person = "Person",
  facets = c("Rater", "Criterion"), score = "Score"
)
summary(out)
p_dq <- plot(out, draw = FALSE)
class(p_dq)

```

---

describe\_mfrm\_data      *Summarize MFRM input data (TAM-style descriptive snapshot)*

---

**Description**

Summarize MFRM input data (TAM-style descriptive snapshot)

**Usage**

```

describe_mfrm_data(
  data,
  person,
  facets,
  score,
  weight = NULL,
  rating_min = NULL,
  rating_max = NULL,
  keep_original = FALSE,
  include_person_facet = FALSE,
  include_agreement = TRUE,
  rater_facet = NULL,
  context_facets = NULL,
  agreement_top_n = NULL
)

```

**Arguments**

data	A data.frame in long format (one row per rating event).
person	Column name for person IDs.
facets	Character vector of facet column names.
score	Column name for observed score.
weight	Optional weight/frequency column name.
rating_min	Optional minimum category value.
rating_max	Optional maximum category value.

keep_original	Keep original category values.
include_person_facet	If TRUE, include person-level rows in facet_level_summary.
include_agreement	If TRUE, include an observed-score inter-rater agreement bundle (summary/pairs/settings) in the output.
rater_facet	Optional rater facet name used for agreement summaries. If NULL, inferred from facet names.
context_facets	Optional facets used to define matched contexts for agreement. If NULL, all remaining facets (including Person) are used.
agreement_top_n	Optional maximum number of agreement pair rows.

### Details

This function provides a compact descriptive bundle similar to the pre-fit summaries commonly checked in TAM workflows: sample size, score distribution, per-facet coverage, and linkage counts. `psych::describe()` is used for numeric descriptives of score and weight.

#### Key data-quality checks to perform before fitting:

- *Sparse categories*: any score category with fewer than 10 weighted observations may produce unstable threshold estimates (Linacre, 2002). Consider collapsing adjacent categories.
- *Unlinked elements*: if a facet level has zero overlap with one or more levels of another facet, the design is disconnected and parameters cannot be placed on a common scale. Check `linkage_summary` for low connectivity.
- *Extreme scores*: persons or facet levels with all-minimum or all-maximum scores yield infinite logit estimates under JML; they are handled via Bayesian shrinkage under MML.

### Value

A list of class `mfrm_data_description` with:

- `overview`: one-row run-level summary
- `missing_by_column`: missing counts in selected input columns
- `score_descriptives`: output from `psych::describe()` for score
- `weight_descriptives`: output from `psych::describe()` for weight
- `score_distribution`: weighted and raw score frequencies
- `facet_level_summary`: per-level usage and score summaries
- `linkage_summary`: person-facet connectivity diagnostics
- `agreement`: observed-score inter-rater agreement bundle

## Interpreting output

Recommended order:

- `overview`: confirms sample size, facet count, and category span. The `MinWeightedN` column shows the smallest weighted observation count across all facet levels; values below 30 may lead to unstable parameter estimates.
- `missing_by_column`: identifies immediate data-quality risks. Any non-zero count warrants investigation before fitting.
- `score_distribution`: checks sparse/unused score categories. Balanced usage across categories is ideal; heavily skewed distributions may compress the measurement range.
- `facet_level_summary` and `linkage_summary`: checks per-level support and person-facet connectivity. Low linkage ratios indicate sparse or disconnected design blocks.
- `agreement`: optional observed inter-rater consistency summary (exact agreement, correlation, mean differences per rater pair).

## Typical workflow

1. Run `describe_mfrm_data()` on long-format input.
2. Review `summary(ds)` and `plot(ds, ...)`.
3. Resolve missingness/sparsity issues before `fit_mfrm()`.

## See Also

`fit_mfrm()`, `audit_mfrm_anchors()`

## Examples

```
toy <- load_mfrmr_data("example_core")
ds <- describe_mfrm_data(
  data = toy,
  person = "Person",
  facets = c("Rater", "Criterion"),
  score = "Score"
)
s_ds <- summary(ds)
s_ds$overview
p_ds <- plot(ds, draw = FALSE)
class(p_ds)
```

---

`detect_anchor_drift`     *Detect anchor drift across multiple calibrations*

---

## Description

Compares facet estimates across two or more calibration waves to identify elements whose difficulty/severity has shifted beyond acceptable thresholds. Useful for monitoring rater drift over time or checking the stability of item banks.

**Usage**

```

detect_anchor_drift(
  fits,
  facets = NULL,
  drift_threshold = 0.5,
  flag_se_ratio = 2,
  reference = 1L,
  include_person = FALSE
)

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

## S3 method for class 'mfrm_anchor_drift'
summary(object, ...)

## S3 method for class 'summary.mfrm_anchor_drift'
print(x, ...)

```

**Arguments**

<code>fits</code>	Named list of <code>mfrm_fit</code> objects (e.g., <code>list(Year1 = fit1, Year2 = fit2)</code> ).
<code>facets</code>	Character vector of facets to compare (default: all non-Person facets).
<code>drift_threshold</code>	Absolute drift threshold for flagging (logits, default 0.5).
<code>flag_se_ratio</code>	Drift/SE ratio threshold for flagging (default 2.0).
<code>reference</code>	Index or name of the reference fit (default: first).
<code>include_person</code>	Include person estimates in comparison.
<code>x</code>	An <code>mfrm_anchor_drift</code> object.
<code>...</code>	Ignored.
<code>object</code>	An <code>mfrm_anchor_drift</code> object (for summary).

**Details**

For each non-reference wave, the function extracts facet-level estimates using `make_anchor_table()` and computes the element-by-element difference against the reference wave. Standard errors are obtained from `diagnose_mfrm()` applied to each fit. Only elements common to both the reference and a comparison wave are included. Before reporting drift, the function removes the weighted common-element link offset between the two waves so that `Drift` represents residual instability rather than the overall shift between calibrations. The function also records how many common elements survive the screening step within each linking facet and treats fewer than 5 retained common elements per facet as thin support.

An element is **flagged** when either condition is met:

$$|\Delta_e| > \text{drift\_threshold}$$

$$|\Delta_e/SE_{\Delta_e}| > \text{flag\_se\_ratio}$$

The dual-criterion approach guards against flagging elements with large but imprecise estimates, and against missing small but precisely estimated shifts.

When `facets` is `NULL`, all non-Person facets are compared. Providing a subset (e.g., `facets = "Criterion"`) restricts comparison to those facets only.

## Value

Object of class `mfrm_anchor_drift` with components:

**drift\_table** Tibble of element-level drift statistics.

**summary** Drift summary aggregated by facet and wave.

**common\_elements** Tibble of pairwise common-element counts.

**common\_by\_facet** Tibble of retained common-element counts by facet.

**config** List of analysis configuration.

## Which function should I use?

- Use `anchor_to_baseline()` when your starting point is raw new data plus a single baseline fit.
- Use `detect_anchor_drift()` when you already have multiple fitted waves and want a reference-versus-wave comparison.
- Use `build_equating_chain()` when the waves form a sequence and you need cumulative linking offsets.

## Interpreting output

- `$drift_table`: one row per element x wave combination, with columns `Facet`, `Level`, `Wave`, `Ref_Est`, `Wave_Est`, `LinkOffset`, `Drift`, `SE_Ref`, `SE_Wave`, `SE`, `Drift_SE_Ratio`, `LinkSupportAdequate`, and `Flag`. Large drift signals instability after alignment to the common-element link.
- `$summary`: aggregated statistics by facet and wave: number of elements, mean/max absolute drift, and count of flagged elements.
- `$common_elements`: pairwise common-element counts in tidy table form. Small overlap weakens the comparison and results should be interpreted cautiously.
- `$common_by_facet`: retained common-element counts by linking facet for each reference-versus-wave comparison. `LinkSupportAdequate = FALSE` means the link rests on fewer than 5 retained common elements in at least one facet.
- `$config`: records the analysis parameters for reproducibility.
- A practical reading order is `summary(drift)` first, then `drift$drift_table`, then `drift$common_by_facet` if overlap looks thin.

**Typical workflow**

1. Fit separate models for each administration wave.
2. Combine into a named list: `fits <- list(Spring = fit_s, Fall = fit_f)`.
3. Call `drift <- detect_anchor_drift(fits)`.
4. Review `summary(drift)` and `plot_anchor_drift(drift)`.
5. Flagged elements may need to be removed from anchor sets or investigated for substantive causes (e.g., rater re-training).

**See Also**

[anchor\\_to\\_baseline\(\)](#), [build\\_equating\\_chain\(\)](#), [make\\_anchor\\_table\(\)](#), [plot\\_anchor\\_drift\(\)](#), [mfrmr\\_linking\\_and\\_dff](#)

**Examples**

```
d1 <- load_mfrmr_data("study1")
d2 <- load_mfrmr_data("study2")
fit1 <- fit_mfrmr(d1, "Person", c("Rater", "Criterion"), "Score",
  method = "JML", maxit = 15)
fit2 <- fit_mfrmr(d2, "Person", c("Rater", "Criterion"), "Score",
  method = "JML", maxit = 15)
drift <- detect_anchor_drift(list(Wave1 = fit1, Wave2 = fit2))
summary(drift)
head(drift$drift_table[, c("Facet", "Level", "Wave", "Drift", "Flag")])
drift$common_elements
```

---

diagnose\_mfrm

---

*Compute diagnostics for an mfrm\_fit object*


---

**Description**

Compute diagnostics for an `mfrm_fit` object

**Usage**

```
diagnose_mfrm(
  fit,
  interaction_pairs = NULL,
  top_n_interactions = 20,
  whexact = FALSE,
  residual_pca = c("none", "overall", "facet", "both"),
  pca_max_factors = 10L
)
```

### Arguments

<code>fit</code>	Output from <code>fit_mfrm()</code> .
<code>interaction_pairs</code>	Optional list of facet pairs.
<code>top_n_interactions</code>	Number of top interactions.
<code>whexact</code>	Use exact ZSTD transformation.
<code>residual_pca</code>	Residual PCA mode: "none", "overall", "facet", or "both".
<code>pca_max_factors</code>	Maximum number of PCA factors to retain per matrix.

### Details

This function computes a diagnostic bundle used by downstream reporting. It calculates element-level fit statistics, approximate facet separation/reliability summaries, residual-based QC diagnostics, and optionally residual PCA for exploratory residual-structure screening.

#### Key fit statistics computed for each element:

- **Infit MnSq**: information-weighted mean-square residual; sensitive to on-target misfitting patterns. Expected value = 1.0.
- **Outfit MnSq**: unweighted mean-square residual; sensitive to off-target outliers. Expected value = 1.0.
- **ZSTD**: Wilson-Hilferty cube-root transformation of MnSq to an approximate standard normal deviate.
- **PTMEA**: point-measure correlation (item-rest correlation in MFRM context); positive values confirm alignment with the latent trait.

#### Misfit flagging guidelines (Bond & Fox, 2015):

- MnSq < 0.5: overfit (too predictable; may inflate reliability)
- MnSq 0.5–1.5: productive for measurement
- MnSq > 1.5: underfit (noise degrades measurement)
- $|ZSTD| > 2$ : statistically significant misfit (5\

When Infit and Outfit disagree, Infit is generally more informative because it downweights extreme observations. Large Outfit with acceptable Infit typically indicates a few outlying responses rather than systematic misfit.

`interaction_pairs` controls which facet interactions are summarized. Each element can be:

- a length-2 character vector such as `c("Rater", "Criterion")`, or
- omitted (NULL) to let the function select top interactions automatically.

Residual PCA behavior:

- "none": skip PCA (fastest; recommended for initial exploration)
- "overall": compute overall residual PCA across all facets

- "facet": compute facet-specific residual PCA for each facet
- "both": compute both overall and facet-specific PCA

Overall PCA examines the person  $\times$  combined-facet residual matrix; facet-specific PCA examines person  $\times$  facet-level matrices. These summaries are exploratory screens for residual structure, not standalone proofs for or against unidimensionality. Facet-specific PCA can help localise where a stronger residual signal is concentrated.

## Value

An object of class `mfrm_diagnostics` including:

- `obs`: observed/expected/residual-level table
- `measures`: facet/person fit table (Infit, Outfit, ZSTD, PTMEA)
- `overall_fit`: overall fit summary
- `fit`: element-level fit diagnostics
- `reliability`: facet-level model/real separation and reliability
- `precision_profile`: one-row summary of the active precision tier and its recommended use
- `precision_audit`: package-native checks for SE, CI, and reliability
- `facet_precision`: facet-level precision summary by distribution basis and SE mode
- `facets_chisq`: fixed/random facet variability summary
- `interactions`: top interaction diagnostics
- `interrater`: inter-rater agreement bundle (`summary`, `pairs`) including agreement and rater-severity spread indices
- `unexpected`: unexpected-response bundle
- `fair_average`: adjusted-score reference bundle
- `displacement`: displacement diagnostics bundle
- `approximation_notes`: method notes for SE/CI/reliability summaries
- `residual_pca_overall`: optional overall PCA object
- `residual_pca_by_facet`: optional facet PCA objects

## Reading key components

Practical interpretation often starts with:

- `overall_fit`: global infit/outfit and degrees of freedom.
- `reliability`: facet-level model/real separation and reliability. MML uses model-based ModelSE values where available; JML keeps these quantities as exploratory approximations.
- `fit`: element-level misfit scan (Infit, Outfit, ZSTD).
- `unexpected`, `fair_average`, `displacement`: targeted QC bundles.
- `approximation_notes`: method notes for SE/CI/reliability summaries.

### Interpreting output

Start with `overall_fit` and `reliability`, then move to element-level diagnostics (`fit`) and targeted bundles (`unexpected`, `fair_average`, `displacement`, `interrater`, `facets_chisq`).

Consistent signals across multiple components are typically more robust than a single isolated warning. For example, an element flagged for both high `Outfit` and high `displacement` is more concerning than one flagged on a single criterion.

`SE` is kept as a compatibility alias for `ModelSE`. `RealSE` is a fit-adjusted companion defined as  $\text{ModelSE} * \sqrt{\max(\text{Infit}, 1)}$ . Reliability tables report model and fit-adjusted bounds from observed variance, error variance, and true variance; JML entries should still be treated as exploratory.

### Typical workflow

1. Run `diagnose_mfrm(fit, residual_pca = "none")` for baseline diagnostics.
2. Inspect `summary(diag)` and targeted tables/plots.
3. If needed, rerun with residual PCA ("`overall`" or "`both`").

### See Also

[fit\\_mfrm\(\)](#), [analyze\\_residual\\_pca\(\)](#), [build\\_visual\\_summaries\(\)](#), [mfrmr\\_visual\\_diagnostics](#), [mfrmr\\_reporting\\_and\\_apa](#)

### Examples

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
s_diag <- summary(diag)
s_diag$overview[, c("Observations", "Facets", "Categories")]
p_qc <- plot_qc_dashboard(fit, diagnostics = diag, draw = FALSE)
class(p_qc)

# Optional: include residual PCA in the diagnostic bundle
diag_pca <- diagnose_mfrm(fit, residual_pca = "overall")
pca <- analyze_residual_pca(diag_pca, mode = "overall")
head(pca$overall_table)

# Reporting route:
prec <- precision_audit_report(fit, diagnostics = diag)
summary(prec)
```

---

`dif_interaction_table` *Compute interaction table between a facet and a grouping variable*

---

### Description

Produces a cell-level interaction table showing Obs-Exp differences, standardized residuals, and screening statistics for each facet-level x group-value cell.

**Usage**

```

dif_interaction_table(
  fit,
  diagnostics,
  facet,
  group,
  data = NULL,
  min_obs = 10,
  p_adjust = "holm",
  abs_t_warn = 2,
  abs_bias_warn = 0.5
)

```

**Arguments**

fit	Output from <code>fit_mfrm()</code> .
diagnostics	Output from <code>diagnose_mfrm()</code> .
facet	Character scalar naming the facet.
group	Character scalar naming the grouping column.
data	Optional data frame with the group column. If NULL (default), the data stored in <code>fit\$prep\$data</code> is used, but it must contain the group column.
min_obs	Minimum observations per cell. Cells with fewer than this many observations are flagged as sparse and their test statistics set to NA. Default 10.
p_adjust	P-value adjustment method, passed to <code>stats::p.adjust()</code> . Default "holm".
abs_t_warn	Threshold for flagging cells by absolute t-value. Default 2.
abs_bias_warn	Threshold for flagging cells by absolute Obs-Exp average (in logits). Default 0.5.

**Details**

This function uses the fitted model's observation-level residuals (from the internal `compute_obs_table()` function) rather than re-estimating the model. For each facet-level x group-value cell, it computes:

- N: number of observations in the cell
- ObsScore: sum of observed scores
- ExpScore: sum of expected scores
- ObsExpAvg: mean observed-minus-expected difference
- Var\_sum: sum of model variances
- StdResidual:  $(\text{ObsScore} - \text{ExpScore}) / \sqrt{\text{Var\_sum}}$
- t: approximate t-statistic (equal to StdResidual)
- df: N - 1
- p\_value: two-tailed p-value from the t-distribution

**Value**

Object of class `mfrm_dif_interaction` with:

- `table`: tibble with per-cell statistics and flags.
- `summary`: tibble summarizing flagged and sparse cell counts.
- `config`: list of analysis parameters.

**When to use this instead of `analyze_dff()`**

Use `dif_interaction_table()` when you want cell-level screening for a single facet-by-group table. Use `analyze_dff()` when you want group-pair contrasts summarized into differential-functioning effect sizes and method-appropriate classifications.

**Further guidance**

For plot selection and follow-up diagnostics, see [mfrmr\\_visual\\_diagnostics](#).

**Interpreting output**

- `$table`: the full interaction table with one row per cell.
- `$summary`: overview counts of flagged and sparse cells.
- `$config`: analysis configuration parameters.
- Cells with  $|t| > \text{abs\_t\_warn}$  or  $|\text{ObsExpAvg}| > \text{abs\_bias\_warn}$  are flagged in the `flag_t` and `flag_bias` columns.
- Sparse cells ( $N < \text{min\_obs}$ ) have `sparse = TRUE` and NA statistics.

**Typical workflow**

1. Fit a model with `fit_mfrm()`.
2. Run `dif_interaction_table(fit, diag, facet = "Rater", group = "Gender", data = df)`.
3. Inspect `$table` for flagged cells.
4. Visualize with `plot_dif_heatmap()`.

**See Also**

[analyze\\_dff\(\)](#), [analyze\\_dif\(\)](#), [plot\\_dif\\_heatmap\(\)](#), [dif\\_report\(\)](#), [estimate\\_bias\(\)](#)

**Examples**

```
toy <- load_mfrmr_data("example_bias")

fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score",
               method = "JML", model = "RSM", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
int <- dif_interaction_table(fit, diag, facet = "Rater",
                           group = "Group", data = toy, min_obs = 2)

int$summary
head(int$table[, c("Level", "GroupValue", "ObsExpAvg", "flag_bias")])
```

---

 dif\_report

*Generate a differential-functioning interpretation report*


---

## Description

Produces APA-style narrative text interpreting the results of a differential-functioning analysis or interaction table. For method = "refit", the report summarises the number of facet levels classified as negligible (A), moderate (B), and large (C). For method = "residual", it summarises screening-positive results, lists the specific levels and their direction, and includes a caveat about the distinction between construct-relevant variation and measurement bias.

## Usage

```
dif_report(dif_result, ...)
```

## Arguments

dif_result	Output from <code>analyze_dff()</code> / <code>analyze_dif()</code> (class <code>mfrm_dff</code> with compatibility class <code>mfrm_dif</code> ) or <code>dif_interaction_table()</code> (class <code>mfrm_dif_interaction</code> ).
...	Currently unused; reserved for future extensions.

## Details

When `dif_result` is an `mfrm_dff/mfrm_dif` object, the report is based on the pairwise differential-functioning contrasts in `$dif_table`. When it is an `mfrm_dif_interaction` object, the report uses the cell-level statistics and flags from `$table`.

For method = "refit", ETS-style magnitude labels are used only when subgroup calibrations were successfully linked back to a common baseline scale; otherwise the report labels those contrasts as unclassified because the refit difference is descriptive rather than comparable on a linked logit scale. For method = "residual", the report describes screening-positive versus screening-negative contrasts instead of applying ETS labels.

## Value

Object of class `mfrm_dif_report` with `narrative`, `counts`, `large_dif`, and `config`.

## Interpreting output

- `$narrative`: character scalar with the full narrative text.
- `$counts`: named integer vector of method-appropriate counts.
- `$large_dif`: tibble of large ETS results (method = "refit") or screening-positive contrasts/cells (method = "residual").
- `$config`: analysis configuration inherited from the input.

**Typical workflow**

1. Run `analyze_dff()` / `analyze_dif()` or `dif_interaction_table()`.
2. Pass the result to `dif_report()`.
3. Print the report or extract `$narrative` for inclusion in a manuscript.

**See Also**

`analyze_dff()`, `analyze_dif()`, `dif_interaction_table()`, `plot_dif_heatmap()`, `build_apa_outputs()`

**Examples**

```
toy <- load_mfrmr_data("example_bias")

fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score",
               method = "JML", model = "RSM", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
dif <- analyze_dff(fit, diag, facet = "Rater", group = "Group", data = toy)
rpt <- dif_report(dif)
cat(rpt$narrative)
```

---

displacement\_table      *Compute displacement diagnostics for facet levels*

---

**Description**

Compute displacement diagnostics for facet levels

**Usage**

```
displacement_table(
  fit,
  diagnostics = NULL,
  facets = NULL,
  anchored_only = FALSE,
  abs_displacement_warn = 0.5,
  abs_t_warn = 2,
  top_n = NULL
)
```

**Arguments**

<code>fit</code>	Output from <code>fit_mfrm()</code> .
<code>diagnostics</code>	Optional output from <code>diagnose_mfrm()</code> .
<code>facets</code>	Optional subset of facets.
<code>anchored_only</code>	If TRUE, keep only directly/group anchored levels.

abs_displacement_warn	Absolute displacement warning threshold.
abs_t_warn	Absolute displacement t-value warning threshold.
top_n	Optional maximum number of rows to keep after sorting.

### Details

Displacement is computed as a one-step Newton update:  $\text{sum}(\text{residual}) / \text{sum}(\text{information})$  for each facet level. This approximates how much a level would move if constraints were relaxed.

### Value

A named list with:

- `table`: displacement diagnostics by level
- `summary`: one-row summary
- `thresholds`: applied thresholds

### Interpreting output

- `table`: level-wise displacement and flag indicators.
- `summary`: count/share of flagged levels.
- `thresholds`: displacement and t-value cutoffs.

Large absolute displacement in anchored levels suggests potential instability in anchor assumptions.

### Typical workflow

1. Run `displacement_table(fit, anchored_only = TRUE)` for anchor checks.
2. Inspect `summary(dispatch)` then detailed rows.
3. Visualize with `plot_displacement()`.

### Output columns

The table data.frame contains:

**Facet, Level** Facet name and element label.

**Displacement** One-step Newton displacement estimate (logits).

**DisplacementSE** Standard error of the displacement.

**DisplacementT** Displacement / SE ratio.

**Estimate, SE** Current measure estimate and its standard error.

**N** Number of observations involving this level.

**AnchorValue, AnchorStatus, AnchorType** Anchor metadata.

**Flag** Logical; TRUE when displacement exceeds thresholds.

### See Also

[diagnose\\_mfrm\(\)](#), [unexpected\\_response\\_table\(\)](#), [fair\\_average\\_table\(\)](#)

## Examples

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
disp <- displacement_table(fit, anchored_only = FALSE)
summary(disp)
p_disp <- plot(disp, draw = FALSE)
class(p_disp)
```

---

ej2021\_data

*Simulated MFRM datasets based on Eckes and Jin (2021)*

---

## Description

Synthetic many-facet rating datasets in long format. All datasets include one row per observed rating.

## Format

A data.frame with 5 columns:

**Study** Study label ("Study1" or "Study2").

**Person** Person/respondent identifier.

**Rater** Rater identifier.

**Criterion** Criterion facet label.

**Score** Observed category score.

## Details

Available data objects:

- mfrmr\_example\_core
- mfrmr\_example\_bias
- ej2021\_study1
- ej2021\_study2
- ej2021\_combined
- ej2021\_study1\_itercal
- ej2021\_study2\_itercal
- ej2021\_combined\_itercal

Naming convention:

- study1 / study2: separate simulation studies
- combined: row-bind of study1 and study2
- \_itercal: iterative-calibration variant

Use `load_mfrmr_data()` for programmatic selection by key.

## Data dimensions

Dataset	Rows	Persons	Raters	Criteria
study1	1842	307	18	3
study2	3287	206	12	9
combined	5129	307	18	12
study1_itercal	1842	307	18	3
study2_itercal	3341	206	12	9
combined_itercal	5183	307	18	12

Score range: 1–4 (four-category rating scale).

## Simulation design

Person ability is drawn from  $N(0, 1)$ . Rater severity effects span approximately  $-0.5$  to  $+0.5$  logits. Criterion difficulty effects span approximately  $-0.3$  to  $+0.3$  logits. Scores are generated from the resulting linear predictor plus Gaussian noise, then discretized into four categories. The `_itercal` variants use a second iteration of calibrated rater severity parameters.

## Interpreting output

Each dataset is already in long format and can be passed directly to `fit_mfrm()` after confirming column-role mapping.

## Typical workflow

1. Inspect available datasets with `list_mfrm_data()`.
2. Load one dataset using `load_mfrm_data()`.
3. Fit and diagnose with `fit_mfrm()` and `diagnose_mfrm()`.

## Source

Simulated for this package with design settings informed by Eckes and Jin (2021).

## Examples

```
data("ej2021_study1", package = "mfrm")
head(ej2021_study1)
table(ej2021_study1$Study)
```

---

estimate\_all\_bias      *Estimate bias across multiple facet pairs*

---

## Description

Estimate bias across multiple facet pairs

## Usage

```
estimate_all_bias(
  fit,
  diagnostics = NULL,
  pairs = NULL,
  include_person = FALSE,
  drop_empty = TRUE,
  keep_errors = TRUE,
  max_abs = 10,
  omit_extreme = TRUE,
  max_iter = 4,
  tol = 0.001
)
```

## Arguments

fit	Output from <a href="#">fit_mfrm()</a> .
diagnostics	Optional output from <a href="#">diagnose_mfrm()</a> . When NULL, diagnostics are computed with <code>residual_pca = "none"</code> .
pairs	Optional list of facet specifications. Each element should be a character vector of length 2 or more, for example <code>list(c("Rater", "Criterion"), c("Task", "Criterion"))</code> . When NULL, all 2-way combinations of modeled facets are used.
include_person	If TRUE and <code>pairs = NULL</code> , include "Person" in the automatically generated pair set.
drop_empty	If TRUE, omit empty bias tables from <code>by_pair</code> while still recording them in the summary table.
keep_errors	If TRUE, retain per-pair error rows in the returned errors table instead of failing the whole batch.
max_abs	Passed to <a href="#">estimate_bias()</a> .
omit_extreme	Passed to <a href="#">estimate_bias()</a> .
max_iter	Passed to <a href="#">estimate_bias()</a> .
tol	Passed to <a href="#">estimate_bias()</a> .

## Details

This function orchestrates repeated calls to `estimate_bias()` across multiple facet pairs and returns a consolidated bundle.

**Bias/interaction** in MFRM refers to a systematic departure from the additive model for a specific combination of facet elements (e.g., a particular rater is unexpectedly harsh on a particular criterion). See `estimate_bias()` for the mathematical formulation.

When `pairs = NULL`, the function builds all 2-way combinations of modelled facets automatically. For a model with facets Rater, Criterion, and Task, this yields Rater×Criterion, Rater×Task, and Criterion×Task.

The summary table aggregates results across pairs:

- Rows: number of interaction cells estimated
- Significant: count of cells with  $|t| \geq 2$
- MeanAbsBias: average absolute bias magnitude (logits)

Per-pair failures (e.g., insufficient data for a sparse pair) are captured in errors rather than stopping the entire batch.

## Value

A named list with class `mfrm_bias_collection`.

## Output

The returned object is a bundle-like list with class `mfrm_bias_collection` and components such as:

- `summary`: one row per requested interaction
- `by_pair`: named list of successful `estimate_bias()` outputs
- `errors`: per-pair error log
- `settings`: resolved execution settings
- `primary`: first successful bias bundle, useful for downstream helpers

## Typical workflow

1. Fit with `fit_mfrm()` and diagnose with `diagnose_mfrm()`.
2. Run `estimate_all_bias()` to compute app-style multi-pair interactions.
3. Pass the resulting `by_pair` list into `reporting_checklist()` or `facet_quality_dashboard()`.

## See Also

`estimate_bias()`, `reporting_checklist()`, `facet_quality_dashboard()`

**Examples**

```

toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score",
               method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
bias_all <- estimate_all_bias(fit, diagnostics = diag)
bias_all$summary[, c("Interaction", "Rows", "Significant")]

```

estimate\_bias

*Estimate legacy-compatible bias/interaction terms iteratively***Description**

Estimate legacy-compatible bias/interaction terms iteratively

**Usage**

```

estimate_bias(
  fit,
  diagnostics,
  facet_a = NULL,
  facet_b = NULL,
  interaction_facets = NULL,
  max_abs = 10,
  omit_extreme = TRUE,
  max_iter = 4,
  tol = 0.001
)

```

**Arguments**

fit	Output from <code>fit_mfrm()</code> .
diagnostics	Output from <code>diagnose_mfrm()</code> .
facet_a	First facet name.
facet_b	Second facet name.
interaction_facets	Character vector of two or more facets to model as one interaction effect. When supplied, this takes precedence over <code>facet_a/facet_b</code> .
max_abs	Bound for absolute bias size.
omit_extreme	Omit extreme-only elements.
max_iter	Iteration cap.
tol	Convergence tolerance.

## Details

**Bias (interaction) in MFRM** refers to a systematic departure from the additive model: a specific rater-criterion (or higher-order) combination produces scores that are consistently higher or lower than predicted by the main effects alone. For example, Rater A might be unexpectedly harsh on Criterion 2 despite being lenient overall.

Mathematically, the bias term  $b_{jc}$  for rater  $j$  on criterion  $c$  modifies the linear predictor:

$$\eta_{njc} = \theta_n - \delta_j - \beta_c - b_{jc}$$

The function estimates  $b_{jc}$  from the residuals of the fitted (additive) model using iterative recalibration in a legacy-compatible style (Myford & Wolfe, 2003, 2004):

$$b_{jc} = \frac{\sum_n (X_{njc} - E_{njc})}{\sum_n \text{Var}_{njc}}$$

Each iteration updates expected scores using the current bias estimates, then re-computes the bias. Convergence is reached when the maximum absolute change in bias estimates falls below `tol`.

- For two-way mode, use `facet_a` and `facet_b` (or `interaction_facets` with length 2).
- For higher-order mode, provide `interaction_facets` with length  $\geq 3$ .

## Value

An object of class `mfrm_bias` with:

- `table`: interaction rows with effect size, SE, screening t/p metadata, reporting-use flags, and fit columns
- `summary`: compact summary statistics
- `chi_sq`: fixed-effect chi-square style screening summary
- `facet_a`, `facet_b`: first two analyzed facet names (legacy compatibility)
- `interaction_facets`, `interaction_order`, `interaction_mode`: full interaction metadata
- `iteration`: iteration history/metadata

## What this screening means

`estimate_bias()` summarizes interaction departures from the additive MFRM. It is best read as a targeted screening tool for potentially noteworthy cells or facet combinations that may merit substantive review.

## What this screening does not justify

- `t` and `Prob.` are screening metrics, not formal inferential quantities.
- A flagged interaction cell is not, by itself, proof of rater bias or construct-irrelevant variance.
- Non-flagged cells should not be over-read as evidence that interaction effects are absent.

### Interpreting output

Use summary for global magnitude, then inspect table for cell-level interaction effects.

Prioritize rows with:

- larger |Bias Size| (effect on logit scale; > 0.5 logits is typically noteworthy, > 1.0 is large)
- larger |t| among the screening metrics (|t| ≥ 2 suggests a screen-positive interaction cell)
- smaller Prob. among the screening metrics

A positive Obs-Exp Average means the cell produced *higher* scores than the additive model predicts (unexpected leniency); negative means unexpected harshness.

iteration helps verify whether iterative recalibration stabilized. If the maximum change on the final iteration is still above tol, consider increasing max\_iter.

### Typical workflow

1. Fit and diagnose model.
2. Run estimate\_bias(...) for target interaction facets.
3. Review summary(bias) and bias\$table.
4. Visualize/report via plot\_bias\_interaction() and build\_fixed\_reports().

### Interpreting key output columns

In bias\$table, the most-used columns are:

- Bias Size: estimated interaction effect  $b_{jc}$  (logit scale)
- t and Prob.: screening metrics, not formal inferential quantities
- Obs-Exp Average: direction and practical size of observed-vs-expected gap on the raw-score metric

The chi\_sq element provides a fixed-effect heterogeneity screen across all interaction cells.

### Recommended next step

Use plot\_bias\_interaction() to inspect the flagged cells visually, then integrate the result with DFF, linking, or substantive scoring review before making formal claims about fairness or invariance.

### See Also

build\_fixed\_reports(), build\_apa\_outputs()

### Examples

```
toy <- load_mfrmr_data("example_bias")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
bias <- estimate_bias(fit, diag, facet_a = "Rater", facet_b = "Criterion", max_iter = 2)
summary(bias)
p_bias <- plot_bias_interaction(bias, draw = FALSE)
class(p_bias)
```

---

`estimation_iteration_report`*Build an estimation-iteration report (preferred alias)*

---

**Description**

Build an estimation-iteration report (preferred alias)

**Usage**

```
estimation_iteration_report(  
  fit,  
  max_iter = 20,  
  reltol = NULL,  
  include_prox = TRUE,  
  include_fixed = FALSE  
)
```

**Arguments**

<code>fit</code>	Output from <code>fit_mfrm()</code> .
<code>max_iter</code>	Maximum replay iterations (excluding optional initial row).
<code>reltol</code>	Stopping tolerance for replayed max-logit change.
<code>include_prox</code>	If TRUE, include an initial pseudo-row labeled PROX.
<code>include_fixed</code>	If TRUE, include a legacy-compatible fixed-width text block.

**Details**

`summary(out)` is supported through `summary()`. `plot(out)` is dispatched through `plot()` for class `mfrm_iteration_report` (`type = "residual", "logit_change", "objective"`).

**Value**

A named list with iteration-report components. Class: `mfrm_iteration_report`.

**Interpreting output**

- `iterations`: trajectory of convergence indicators by iteration.
- `summary`: final status and stopping diagnostics.
- optional PROX row: pseudo-initial reference point when enabled.

**Typical workflow**

1. Run `estimation_iteration_report(fit)`.
2. Inspect plateau/stability patterns in `summary/plot`.
3. Adjust optimization settings if convergence looks weak.

**See Also**

[fit\\_mfrm\(\)](#), [specifications\\_report\(\)](#), [data\\_quality\\_report\(\)](#), [mfrm\\_reports\\_and\\_tables](#), [mfrmr\\_compatibility\\_layer](#)

**Examples**

```
toy <- load_mfrm_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
out <- estimation_iteration_report(fit, max_iter = 5)
summary(out)
p_iter <- plot(out, draw = FALSE)
class(p_iter)
```

---

evaluate\_mfrm\_design *Evaluate MFRM design conditions by repeated simulation*

---

**Description**

Evaluate MFRM design conditions by repeated simulation

**Usage**

```
evaluate_mfrm_design(
  n_person = c(30, 50, 100),
  n_rater = c(3, 5),
  n_criterion = c(3, 5),
  raters_per_person = n_rater,
  reps = 10,
  score_levels = 4,
  theta_sd = 1,
  rater_sd = 0.35,
  criterion_sd = 0.25,
  noise_sd = 0,
  step_span = 1.4,
  fit_method = c("JML", "MML"),
  model = c("RSM", "PCM"),
  step_facet = NULL,
  maxit = 25,
  quad_points = 7,
  residual_pca = c("none", "overall", "facet", "both"),
  sim_spec = NULL,
  seed = NULL
)
```

**Arguments**

n_person	Vector of person counts to evaluate.
n_rater	Vector of rater counts to evaluate.
n_criterion	Vector of criterion counts to evaluate.
raters_per_person	Vector of rater assignments per person.
reps	Number of replications per design condition.
score_levels	Number of ordered score categories.
theta_sd	Standard deviation of simulated person measures.
rater_sd	Standard deviation of simulated rater severities.
criterion_sd	Standard deviation of simulated criterion difficulties.
noise_sd	Optional observation-level noise added to the linear predictor.
step_span	Spread of step thresholds on the logit scale.
fit_method	Estimation method passed to <code>fit_mfrm()</code> .
model	Measurement model passed to <code>fit_mfrm()</code> .
step_facet	Step facet passed to <code>fit_mfrm()</code> when <code>model = "PCM"</code> . When left NULL, the function inherits the generator step facet from <code>sim_spec</code> when available and otherwise defaults to "Criterion".
maxit	Maximum iterations passed to <code>fit_mfrm()</code> .
quad_points	Quadrature points for <code>fit_method = "MML"</code> .
residual_pca	Residual PCA mode passed to <code>diagnose_mfrm()</code> .
sim_spec	Optional output from <code>build_mfrm_sim_spec()</code> or <code>extract_mfrm_sim_spec()</code> used as the base data-generating mechanism. When supplied, the design grid still varies <code>n_person</code> , <code>n_rater</code> , <code>n_criterion</code> , and <code>raters_per_person</code> , but latent-spread assumptions, thresholds, and other generator settings come from <code>sim_spec</code> . If <code>sim_spec</code> contains step-facet-specific thresholds, the design grid may not vary the number of levels for that step facet away from the specification.
seed	Optional seed for reproducible replications.

**Details**

This helper runs a compact Monte Carlo design study for common rater-by-item many-facet settings.

For each design condition, the function:

1. generates synthetic data with `simulate_mfrm_data()`
2. fits the requested MFRM with `fit_mfrm()`
3. computes diagnostics with `diagnose_mfrm()`
4. stores recovery and precision summaries by facet

The result is intended for planning questions such as:

- how many raters are needed for stable rater separation?

- how does raters\_per\_person affect severity recovery?
- when do category counts become too sparse for comfortable interpretation?

This is a **parametric simulation study**. It does not take one observed design (for example, 4 raters x 30 persons x 3 criteria) and analytically extrapolate what would happen under a different design (for example, 2 raters x 40 persons x 5 criteria). Instead, you specify a design grid and data-generating assumptions (latent spread, facet spread, thresholds, noise, and scoring structure), and the function repeatedly generates synthetic data under those assumptions.

When you want the simulated conditions to resemble an existing study, use substantive knowledge or estimates from that study to choose `theta_sd`, `rater_sd`, `criterion_sd`, `score_levels`, and related settings before running the design evaluation.

When `sim_spec` is supplied, the function uses it as the explicit data-generating mechanism. This is the recommended route when you want a design study to stay close to a previously fitted run while still varying the candidate sample sizes or rater-assignment counts.

Recovery metrics are reported only when the generator and fitted model target the same facet-parameter contract. In practice this means the same model, and for PCM, the same `step_facet`. When these do not align, recovery fields are set to NA and the output records the reason.

## Value

An object of class `mfrm_design_evaluation` with components:

- `design_grid`: evaluated design conditions
- `results`: facet-level replicate results
- `rep_overview`: run-level status and timing
- `settings`: simulation settings
- `ademp`: simulation-study metadata (aims, DGM, estimands, methods, performance measures)

## Reported metrics

Facet-level simulation results include:

- Separation ( $G = SD_{adj}/RMSE$ ): how many statistically distinct strata the facet resolves.
- Reliability ( $G^2/(1 + G^2)$ ): analogous to Cronbach's  $\alpha$  for the reproducibility of element ordering.
- Strata  $((4G + 1)/3)$ : number of distinguishable groups.
- Mean Infit and Outfit: average fit mean-squares across elements.
- MisfitRate: share of elements with  $|ZSTD| > 2$ .
- SeverityRMSE: root-mean-square error of recovered parameters vs the known truth **after facet-wise mean alignment**, so that the usual Rasch/MFRM location indeterminacy does not inflate recovery error. This quantity is reported only when the generator and fitted model target the same facet-parameter contract.
- SeverityBias: mean signed recovery error after the same alignment; values near zero are expected. This is likewise omitted when the generator/fitted-model contract does not align.

### Interpreting output

Start with `summary(x)$design_summary`, then plot one focal metric at a time (for example rater Separation or criterion SeverityRMSE).

Higher separation/reliability is generally better, whereas lower SeverityRMSE, MeanMisfitRate, and MeanElapsedSec are preferable.

When choosing among designs, look for the point where increasing `n_person` or `raters_per_person` yields diminishing returns in separation and RMSE—this identifies the cost-effective design frontier. ConvergedRuns / reps should be near 1.0; low convergence rates indicate the design is too small for the chosen estimation method.

### References

The simulation logic follows the general Monte Carlo / operating-characteristic framework described by Morris, White, and Crowther (2019) and the ADEMP-oriented planning/reporting guidance summarized for psychology by Siepe et al. (2024). In `mfrmr`, `evaluate_mfrm_design()` is a practical many-facet design-planning wrapper rather than a direct reproduction of one published simulation study.

- Morris, T. P., White, I. R., & Crowther, M. J. (2019). *Using simulation studies to evaluate statistical methods*. *Statistics in Medicine*, 38(11), 2074-2102.
- Siepe, B. S., Bartos, F., Morris, T. P., Boulesteix, A.-L., Heck, D. W., & Pawel, S. (2024). *Simulation studies for methodological research in psychology: A standardized template for planning, preregistration, and reporting*. *Psychological Methods*.

### See Also

[simulate\\_mfrm\\_data\(\)](#), [summary.mfrm\\_design\\_evaluation](#), [plot.mfrm\\_design\\_evaluation](#)

### Examples

```
sim_eval <- evaluate_mfrm_design(
  n_person = c(30, 50),
  n_rater = 4,
  n_criterion = 4,
  raters_per_person = 2,
  reps = 1,
  maxit = 15,
  seed = 123
)
s_eval <- summary(sim_eval)
s_eval$design_summary[, c("Facet", "n_person", "MeanSeparation", "MeanSeverityRMSE")]
p_eval <- plot(sim_eval, facet = "Rater", metric = "separation", x_var = "n_person", draw = FALSE)
names(p_eval)
```

---

`evaluate_mfrm_signal_detection`

*Evaluate DIF power and bias-screening behavior under known simulated signals*

---

**Description**

Evaluate DIF power and bias-screening behavior under known simulated signals

**Usage**

```
evaluate_mfrm_signal_detection(  
  n_person = c(30, 50, 100),  
  n_rater = c(4),  
  n_criterion = c(4),  
  raters_per_person = n_rater,  
  reps = 10,  
  group_levels = c("A", "B"),  
  reference_group = NULL,  
  focal_group = NULL,  
  dif_level = NULL,  
  dif_effect = 0.6,  
  bias_rater = NULL,  
  bias_criterion = NULL,  
  bias_effect = -0.8,  
  score_levels = 4,  
  theta_sd = 1,  
  rater_sd = 0.35,  
  criterion_sd = 0.25,  
  noise_sd = 0,  
  step_span = 1.4,  
  fit_method = c("JML", "MML"),  
  model = c("RSM", "PCM"),  
  step_facet = NULL,  
  maxit = 25,  
  quad_points = 7,  
  residual_pca = c("none", "overall", "facet", "both"),  
  sim_spec = NULL,  
  dif_method = c("residual", "refit"),  
  dif_min_obs = 10,  
  dif_p_adjust = "holm",  
  dif_p_cut = 0.05,  
  dif_abs_cut = 0.43,  
  bias_max_iter = 2,  
  bias_p_cut = 0.05,  
  bias_abs_t = 2,  
  seed = NULL
```

)

**Arguments**

n_person	Vector of person counts to evaluate.
n_rater	Vector of rater counts to evaluate.
n_criterion	Vector of criterion counts to evaluate.
raters_per_person	Vector of rater assignments per person.
reps	Number of replications per design condition.
group_levels	Group labels used for DIF simulation. The first two levels define the default reference and focal groups.
reference_group	Optional reference group label used when extracting the target DIF contrast.
focal_group	Optional focal group label used when extracting the target DIF contrast.
dif_level	Target criterion level for the true DIF effect. Can be an integer index or a criterion label such as "C04". Defaults to the last criterion level in each design.
dif_effect	True DIF effect size added to the focal group on the target criterion.
bias_rater	Target rater level for the true interaction-bias effect. Can be an integer index or a label such as "R04". Defaults to the last rater level in each design.
bias_criterion	Target criterion level for the true interaction-bias effect. Can be an integer index or a criterion label. Defaults to the last criterion level in each design.
bias_effect	True interaction-bias effect added to the target Rater x Criterion cell.
score_levels	Number of ordered score categories.
theta_sd	Standard deviation of simulated person measures.
rater_sd	Standard deviation of simulated rater severities.
criterion_sd	Standard deviation of simulated criterion difficulties.
noise_sd	Optional observation-level noise added to the linear predictor.
step_span	Spread of step thresholds on the logit scale.
fit_method	Estimation method passed to <code>fit_mfrm()</code> .
model	Measurement model passed to <code>fit_mfrm()</code> .
step_facet	Step facet passed to <code>fit_mfrm()</code> when <code>model = "PCM"</code> . When left NULL, the function inherits the generator step facet from <code>sim_spec</code> when available and otherwise defaults to "Criterion".
maxit	Maximum iterations passed to <code>fit_mfrm()</code> .
quad_points	Quadrature points for <code>fit_method = "MML"</code> .
residual_pca	Residual PCA mode passed to <code>diagnose_mfrm()</code> .
sim_spec	Optional output from <code>build_mfrm_sim_spec()</code> or <code>extract_mfrm_sim_spec()</code> used as the base data-generating mechanism. When supplied, the design grid still varies <code>n_person</code> , <code>n_rater</code> , <code>n_criterion</code> , and <code>raters_per_person</code> , but latent spread, thresholds, and other generator settings come from <code>sim_spec</code> . The target DIF and interaction-bias signals specified in this function override any signal tables stored in <code>sim_spec</code> .

dif_method	Differential-functioning method passed to <code>analyze_dff()</code> .
dif_min_obs	Minimum observations per group cell for <code>analyze_dff()</code> .
dif_p_adjust	P-value adjustment method passed to <code>analyze_dff()</code> .
dif_p_cut	P-value cutoff for counting a target DIF detection.
dif_abs_cut	Optional absolute contrast cutoff used when counting a target DIF detection. When omitted, the effective default is 0.43 for <code>dif_method = "refit"</code> and 0 (no additional magnitude cutoff) for <code>dif_method = "residual"</code> .
bias_max_iter	Maximum iterations passed to <code>estimate_bias()</code> .
bias_p_cut	P-value cutoff for counting a target bias screen-positive result.
bias_abs_t	Absolute t cutoff for counting a target bias screen-positive result.
seed	Optional seed for reproducible replications.

## Details

This function performs Monte Carlo design screening for two related tasks: DIF detection via `analyze_dff()` and interaction-bias screening via `estimate_bias()`.

For each design condition (combination of `n_person`, `n_rater`, `n_criterion`, `raters_per_person`), the function:

1. Generates synthetic data with `simulate_mfrm_data()`
2. Injects one known Group  $\times$  Criterion DIF effect (`dif_effect` logits added to the focal group on the target criterion)
3. Injects one known Rater  $\times$  Criterion interaction-bias effect (`bias_effect` logits)
4. Fits and diagnoses the MFRM
5. Runs `analyze_dff()` and `estimate_bias()`
6. Records whether the injected signals were detected or screen-positive

**Detection criteria:** A DIF signal is counted as "detected" when the target contrast has  $p < \text{dif\_p\_cut}$  **and**, when an absolute contrast cutoff is in force,  $|\text{Contrast}| \geq \text{dif\_abs\_cut}$ . For `dif_method = "refit"`, `dif_abs_cut` is interpreted on the logit scale. For `dif_method = "residual"`, the residual-contrast screening result is used and the default is to rely on the significance test alone.

Bias results are different: `estimate_bias()` reports `t` and `Prob.` as screening metrics rather than formal inferential quantities. Here, a bias cell is counted as **screen-positive** only when those screening metrics are available and satisfy  $p < \text{bias\_p\_cut}$  **and**  $|t| \geq \text{bias\_abs\_t}$ .

**Power** is the proportion of replications in which the target signal was correctly detected. For DIF this is a conventional power summary. For bias, the primary summary is `BiasScreenRate`, a screening hit rate rather than formal inferential power.

**False-positive rate** is the proportion of non-target cells that were incorrectly flagged. For DIF this is interpreted in the usual testing sense. For bias, `BiasScreenFalsePositiveRate` is a screening rate and should not be read as a calibrated inferential alpha level.

**Default effect sizes:** `dif_effect = 0.6` logits corresponds to a moderate criterion-linked differential-functioning effect; `bias_effect = -0.8` logits represents a substantial rater-criterion interaction. Adjust these to match the smallest effect size of practical concern for your application.

This is again a **parametric simulation study**. The function does not estimate a new design directly from one observed dataset. Instead, it evaluates detection or screening behavior under user-specified design conditions and known injected signals.

If you want to approximate a real study, choose the design grid and simulation settings so that they reflect the empirical context of interest. For example, you may set `n_person`, `n_rater`, `n_criterion`, `raters_per_person`, and the latent-spread arguments to values motivated by an existing assessment program, then study how operating characteristics change as those design settings vary.

When `sim_spec` is supplied, the function uses it as the explicit data-generating mechanism for the latent spreads, thresholds, and assignment archetype, while still injecting the requested target DIF and bias effects for each design condition.

### Value

An object of class `mfrm_signal_detection` with:

- `design_grid`: evaluated design conditions
- `results`: replicate-level detection results
- `rep_overview`: run-level status and timing
- `settings`: signal-analysis settings
- `ademp`: simulation-study metadata (aims, DGM, estimands, methods, performance measures)

### References

The simulation logic follows the general Monte Carlo / operating-characteristic framework described by Morris, White, and Crowther (2019) and the ADEMP-oriented planning/reporting guidance summarized for psychology by Siepe et al. (2024). In `mfrm`, `evaluate_mfrm_signal_detection()` is a many-facet screening helper specialized to DIF and interaction-bias use cases; it is not a direct implementation of one published many-facet Rasch simulation design.

- Morris, T. P., White, I. R., & Crowther, M. J. (2019). *Using simulation studies to evaluate statistical methods*. *Statistics in Medicine*, 38(11), 2074-2102.
- Siepe, B. S., Bartos, F., Morris, T. P., Boulesteix, A.-L., Heck, D. W., & Pawel, S. (2024). *Simulation studies for methodological research in psychology: A standardized template for planning, preregistration, and reporting*. *Psychological Methods*.

### See Also

[simulate\\_mfrm\\_data\(\)](#), [evaluate\\_mfrm\\_design\(\)](#), [analyze\\_dff\(\)](#), [analyze\\_dif\(\)](#), [estimate\\_bias\(\)](#)

### Examples

```
sig_eval <- suppressWarnings(evaluate_mfrm_signal_detection(
  n_person = 20,
  n_rater = 3,
  n_criterion = 3,
  raters_per_person = 2,
  reps = 1,
```

```

    maxit = 10,
    bias_max_iter = 1,
    seed = 123
  ))
  s_sig <- summary(sig_eval)
  s_sig$detection_summary[, c("n_person", "DIFPower", "BiasScreenRate")]

```

---

 export\_mfrm

*Export MFRM results to CSV files*


---

## Description

Writes tidy CSV files suitable for import into spreadsheet software or further analysis in other tools.

## Usage

```

export_mfrm(
  fit,
  diagnostics = NULL,
  output_dir = ".",
  prefix = "mfrm",
  tables = c("person", "facets", "summary", "steps", "measures"),
  overwrite = FALSE
)

```

## Arguments

fit	Output from <a href="#">fit_mfrm</a> .
diagnostics	Optional output from <a href="#">diagnose_mfrm</a> . When provided, enriches facet estimates with SE, fit statistics, and writes the full measures table.
output_dir	Directory for CSV files. Created if it does not exist.
prefix	Filename prefix (default "mfrm").
tables	Character vector of tables to export. Any subset of "person", "facets", "summary", "steps", "measures". Default exports all available tables.
overwrite	If FALSE (default), refuse to overwrite existing files.

## Value

Invisibly, a data.frame listing written files with columns Table and Path.

## Exported files

```

{prefix}_person_estimates.csv Person ID, Estimate, SD.
{prefix}_facet_estimates.csv Facet, Level, Estimate, and optionally SE, Infit, Outfit, PT-
  MEA when diagnostics supplied.
{prefix}_fit_summary.csv One-row model summary.
{prefix}_step_parameters.csv Step/threshold parameters.
{prefix}_measures.csv Full measures table (requires diagnostics).

```

**Interpreting output**

The returned data.frame tells you exactly which files were written and where. This is convenient for scripted pipelines where the output directory is created on the fly.

**Typical workflow**

1. Fit a model with `fit_mfrm()`.
2. Optionally compute diagnostics with `diagnose_mfrm()` when you want enriched facet or measures exports.
3. Call `export_mfrm(...)` and inspect the returned Path column.

**See Also**

[fit\\_mfrm](#), [diagnose\\_mfrm](#), [as.data.frame.mfrm\\_fit](#)

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score",
               method = "JML", model = "RSM", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
out <- export_mfrm(
  fit,
  diagnostics = diag,
  output_dir = tempdir(),
  prefix = "mfrmr_example",
  overwrite = TRUE
)
out$Table
```

---

export\_mfrm\_bundle      *Export an analysis bundle for sharing or archiving*

---

**Description**

Export an analysis bundle for sharing or archiving

**Usage**

```
export_mfrm_bundle(
  fit,
  diagnostics = NULL,
  bias_results = NULL,
  population_prediction = NULL,
  unit_prediction = NULL,
  plausible_values = NULL,
  output_dir = ".",
```

```

prefix = "mfrmr_bundle",
include = c("core_tables", "checklist", "dashboard", "apa", "anchors", "manifest",
  "visual_summaries", "predictions", "script", "html"),
facet = NULL,
include_person_anchors = FALSE,
overwrite = FALSE,
zip_bundle = FALSE,
zip_name = NULL
)

```

### Arguments

fit	Output from <code>fit_mfrm()</code> or <code>run_mfrm_facets()</code> .
diagnostics	Optional output from <code>diagnose_mfrm()</code> . When NULL, diagnostics are reused from <code>run_mfrm_facets()</code> when available, otherwise computed with <code>residual_pca = "none"</code> (or <code>"both"</code> when visual summaries are requested).
bias_results	Optional output from <code>estimate_bias()</code> or a named list of bias bundles.
population_prediction	Optional output from <code>predict_mfrm_population()</code> .
unit_prediction	Optional output from <code>predict_mfrm_units()</code> .
plausible_values	Optional output from <code>sample_mfrm_plausible_values()</code> .
output_dir	Directory where files will be written.
prefix	File-name prefix.
include	Components to export. Supported values are <code>"core_tables"</code> , <code>"checklist"</code> , <code>"dashboard"</code> , <code>"apa"</code> , <code>"anchors"</code> , <code>"manifest"</code> , <code>"visual_summaries"</code> , <code>"predictions"</code> , <code>"script"</code> , and <code>"html"</code> .
facet	Optional facet for <code>facet_quality_dashboard()</code> .
include_person_anchors	If TRUE, include person measures in the exported anchor table.
overwrite	If FALSE, refuse to overwrite existing files.
zip_bundle	If TRUE, attempt to zip the written files into a single archive using <code>utils::zip()</code> . This is best-effort and may depend on the local R installation.
zip_name	Optional zip-file name. Defaults to <code>"{prefix}_bundle.zip"</code> .

### Details

This function is the package-native counterpart to the app's download bundle. It reuses existing mfrmr helpers instead of reimplementing estimation or diagnostics.

### Value

A named list with class `mfrm_export_bundle`.

## Choosing exports

The include argument lets you assemble a bundle for different audiences:

- "core\_tables" for analysts who mainly want CSV output.
- "manifest" for a compact analysis record.
- "script" for reproducibility and reruns.
- "html" for a light, shareable summary page.
- "visual\_summaries" when you want warning maps or residual PCA summaries to travel with the bundle.

## Recommended presets

Common starting points are:

- minimal tables: `include = c("core_tables", "manifest")`
- reporting bundle: `include = c("core_tables", "checklist", "dashboard", "html")`
- archival bundle: `include = c("core_tables", "manifest", "script", "visual_summaries", "html")`

## Written outputs

Depending on include, the exporter can write:

- core CSV tables via `export_mfrm()`
- checklist CSVs via `reporting_checklist()`
- facet-dashboard CSVs via `facet_quality_dashboard()`
- APA text files via `build_apa_outputs()`
- anchor CSV via `make_anchor_table()`
- manifest CSV/TXT via `build_mfrm_manifest()`
- visual warning/summary artifacts via `build_visual_summaries()`
- prediction/forecast CSVs via `predict_mfrm_population()`, `predict_mfrm_units()`, and `sample_mfrm_plausible_values()`
- a package-native replay script via `build_mfrm_replay_script()`
- a lightweight HTML report that bundles the exported tables/text

## Interpreting output

The returned object reports both high-level bundle status and the exact files written. In practice, `bundle$summary` is the quickest sanity check, while `bundle$written_files` is the file inventory to inspect or hand off to other tools.

**Typical workflow**

1. Fit a model and compute diagnostics once.
2. Decide whether the audience needs tables only, or also a manifest, replay script, and HTML summary.
3. Call `export_mfrm_bundle()` with a dedicated output directory.
4. Inspect `bundle$written_files` or open the generated HTML file.

**See Also**

[build\\_mfrm\\_manifest\(\)](#), [build\\_mfrm\\_replay\\_script\(\)](#), [export\\_mfrm\(\)](#), [reporting\\_checklist\(\)](#)

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score",
              method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
bundle <- export_mfrm_bundle(
  fit,
  diagnostics = diag,
  output_dir = tempdir(),
  prefix = "mfrmr_bundle_example",
  include = c("core_tables", "manifest", "script", "html"),
  overwrite = TRUE
)
bundle$summary[, c("FilesWritten", "HtmlWritten", "ScriptWritten")]
head(bundle$written_files)
```

---

`extract_mfrm_sim_spec` *Derive a simulation specification from a fitted MFRM object*

---

**Description**

Derive a simulation specification from a fitted MFRM object

**Usage**

```
extract_mfrm_sim_spec(
  fit,
  assignment = c("auto", "crossed", "rotating", "resampled", "skeleton"),
  latent_distribution = c("normal", "empirical"),
  source_data = NULL,
  person = NULL,
  group = NULL
)
```

**Arguments**

fit	Output from <code>fit_mfrm()</code> .
assignment	Assignment design to record in the returned specification. Use "resampled" to reuse empirical person-level rater-assignment profiles from the fitted data, or "skeleton" to reuse the observed person-by-facet design skeleton from the fitted data.
latent_distribution	Latent-value generator to record in the returned specification. "normal" stores spread summaries for parametric draws; "empirical" additionally activates centered empirical resampling from the fitted person/rater/criterion estimates.
source_data	Optional original source data used to recover additional non-calibration columns, currently person-level group labels, when building a fit-derived observed response skeleton.
person	Optional person column name in source_data. Defaults to the person column recorded in fit.
group	Optional group column name in source_data to merge into the returned design_skeleton as person-level metadata.

**Details**

`extract_mfrm_sim_spec()` uses a fitted model as a practical starting point for later simulation studies. It extracts:

- design counts from the fitted data
- empirical spread of person and facet estimates
- optional empirical support values for semi-parametric draws
- fitted threshold values
- either a simplified assignment summary ("crossed" / "rotating"), empirical resampled assignment profiles ("resampled"), or an observed response skeleton ("skeleton", optionally carrying Group/Weight)

This is intended as a **fit-derived parametric starting point**, not as a claim that the fitted object perfectly recovers the true data-generating mechanism. Users should review and, if necessary, edit the returned specification before using it for design planning.

If you want to carry person-level group labels into a fit-derived observed response skeleton, provide the original source\_data together with person and group. Group labels are treated as person-level metadata and are checked for one-label-per-person consistency before being merged.

**Value**

An object of class `mfrm_sim_spec`.

**Interpreting output**

The returned object is a simulation specification, not a prediction about one future sample. It captures one convenient approximation to the observed design and estimated spread in the fitted run.

**See Also**

[build\\_mfrm\\_sim\\_spec\(\)](#), [simulate\\_mfrm\\_data\(\)](#)

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 15)
spec <- extract_mfrm_sim_spec(fit, latent_distribution = "empirical")
spec$assignment
spec$model
head(spec$threshold_table)
```

---

facets\_chisq\_table      *Build facet variability diagnostics with fixed/random reference tests*

---

**Description**

Build facet variability diagnostics with fixed/random reference tests

**Usage**

```
facets_chisq_table(
  fit,
  diagnostics = NULL,
  fixed_p_max = 0.05,
  random_p_max = 0.05,
  top_n = NULL
)
```

**Arguments**

<code>fit</code>	Output from <a href="#">fit_mfrm()</a> .
<code>diagnostics</code>	Optional output from <a href="#">diagnose_mfrm()</a> .
<code>fixed_p_max</code>	Warning cutoff for fixed-effect chi-square p-values.
<code>random_p_max</code>	Warning cutoff for random-effect chi-square p-values.
<code>top_n</code>	Optional maximum number of facet rows to keep.

**Details**

This helper summarizes facet-level variability with fixed and random chi-square indices for spread and heterogeneity checks.

**Value**

A named list with:

- `table`: facet-level chi-square diagnostics
- `summary`: one-row summary
- `thresholds`: applied p-value thresholds

### Interpreting output

- **table**: facet-level fixed/random chi-square and p-value flags.
- **summary**: number of significant facets and overall magnitude indicators.
- **thresholds**: p-value criteria used for flagging.

Use this table together with inter-rater and displacement diagnostics to distinguish global facet effects from local anomalies.

### Typical workflow

1. Run `facets_chisq_table(fit, ...)`.
2. Inspect `summary(chi)` then facet rows in `chi$table`.
3. Visualize with `plot_facets_chisq()`.

### Output columns

The table data.frame contains:

**Facet** Facet name.

**Levels** Number of estimated levels in this facet.

**MeanMeasure, SD** Mean and standard deviation of level measures.

**FixedChiSq, FixedDF, FixedProb** Fixed-effect chi-square test (null hypothesis: all levels equal). Significant result means the facet elements differ more than measurement error alone.

**RandomChiSq, RandomDF, RandomProb, RandomVar** Random-effect test (null hypothesis: variation equals that of a random sample from a single population). Significant result suggests systematic heterogeneity beyond sampling variation.

**FixedFlag, RandomFlag** Logical flags for significance.

### See Also

`diagnose_mfrm()`, `interrater_agreement_table()`, `plot_facets_chisq()`

### Examples

```
toy <- load_mfrm_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
chi <- facets_chisq_table(fit)
summary(chi)
p_chi <- plot(chi, draw = FALSE)
class(p_chi)
```

---

 facets\_output\_file\_bundle

*Build a legacy-compatible output-file bundle (GRAPH= / SCORE=)*


---

## Description

Build a legacy-compatible output-file bundle (GRAPH= / SCORE=)

## Usage

```
facets_output_file_bundle(
  fit,
  diagnostics = NULL,
  include = c("graph", "score"),
  theta_range = c(-6, 6),
  theta_points = 241,
  digits = 4,
  include_fixed = FALSE,
  fixed_max_rows = 400,
  write_files = FALSE,
  output_dir = NULL,
  file_prefix = "mfrmr_output",
  overwrite = FALSE
)
```

## Arguments

<code>fit</code>	Output from <code>fit_mfrm()</code> .
<code>diagnostics</code>	Optional output from <code>diagnose_mfrm()</code> (used for score file).
<code>include</code>	Output components to include: "graph" and/or "score".
<code>theta_range</code>	Theta/logit range for graph coordinates.
<code>theta_points</code>	Number of points on the theta grid for graph coordinates.
<code>digits</code>	Rounding digits for numeric fields.
<code>include_fixed</code>	If TRUE, include fixed-width text mirrors of output tables.
<code>fixed_max_rows</code>	Maximum rows shown in fixed-width text blocks.
<code>write_files</code>	If TRUE, write selected outputs to files in <code>output_dir</code> .
<code>output_dir</code>	Output directory used when <code>write_files = TRUE</code> .
<code>file_prefix</code>	Prefix used for output file names.
<code>overwrite</code>	If FALSE, existing output files are not overwritten.

## Details

Legacy-compatible output files often include:

- graph coordinates for Table 8 curves (GRAPH= / Graphfile=), and
- observation-level modeled score lines (SCORE=-style inspection).

This helper returns both as data frames and can optionally write CSV/fixed-width text files to disk. `summary(out)` is supported through `summary()`. `plot(out)` is dispatched through `plot()` for class `mfrm_output_bundle` (type = "graph\_expected", "score\_residuals", "obs\_probability").

## Value

A named list including:

- `graphfile` / `graphfile_syntactic` when "graph" is requested
- `scorefile` when "score" is requested
- `graphfile_fixed` / `scorefile_fixed` when `include_fixed = TRUE`
- `written_files` when `write_files = TRUE`
- `settings`: applied options

## Interpreting output

- `graphfile`: legacy-compatible wide curve coordinates (human-readable labels).
- `graphfile_syntactic`: same curves with syntactic column names for programmatic use.
- `scorefile`: observation-level observed/expected/residual diagnostics.
- `written_files`: audit trail of files produced when `write_files = TRUE`.

For reproducible pipelines, prefer `graphfile_syntactic` and keep `written_files` in run logs.

## Preferred route for new analyses

For new scripts, prefer `category_curves_report()` or `category_structure_report()` for scale outputs, then use `export_mfrm_bundle()` for file handoff. Use `facets_output_file_bundle()` only when a legacy-compatible `graphfile` or `scorefile` contract is required.

## Typical workflow

1. Fit and diagnose model.
2. Generate bundle with `include = c("graph", "score")`.
3. Validate with `summary(out) / plot(out)`.
4. Export with `write_files = TRUE` for reporting handoff.

## See Also

[category\\_curves\\_report\(\)](#), [diagnose\\_mfrm\(\)](#), [unexpected\\_response\\_table\(\)](#), [export\\_mfrm\\_bundle\(\)](#), [mfrm\\_reports\\_and\\_tables](#), [mfrm\\_compatibility\\_layer](#)

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
out <- facets_output_file_bundle(fit, diagnostics = diagnose_mfrm(fit, residual_pca = "none"))
summary(out)
p_out <- plot(out, draw = FALSE)
class(p_out)
```

---

facets\_parity\_report *Build a FACETS compatibility-contract audit*

---

**Description**

Build a FACETS compatibility-contract audit

**Usage**

```
facets_parity_report(
  fit,
  diagnostics = NULL,
  bias_results = NULL,
  branch = c("facets", "original"),
  contract_file = NULL,
  include_metrics = TRUE,
  top_n_missing = 15L
)
```

**Arguments**

<code>fit</code>	Output from <code>fit_mfrm()</code> .
<code>diagnostics</code>	Optional output from <code>diagnose_mfrm()</code> . If omitted, diagnostics are computed internally with <code>residual_pca = "none"</code> .
<code>bias_results</code>	Optional output from <code>estimate_bias()</code> . If omitted and at least two facets exist, a 2-way bias run is computed internally.
<code>branch</code>	Contract branch. "facets" checks legacy-compatible columns. "original" adapts branch-sensitive contracts to the package's compact naming.
<code>contract_file</code>	Optional path to a custom contract CSV.
<code>include_metrics</code>	If TRUE, run additional numerical consistency checks.
<code>top_n_missing</code>	Number of lowest-coverage contract rows to keep in <code>missing_preview</code> .

## Details

This function audits produced report components against a compatibility contract specification (`inst/references/facets_column_contract.csv`) and returns:

- column-level coverage per contract row
- table-level coverage summaries
- optional metric-level consistency checks

It is intended for compatibility-layer QA and regression auditing. It does not establish external validity or software equivalence beyond the specific schema/metric contract encoded in the audit file.

Coverage interpretation in overall:

- `MeanColumnCoverage` and `MinColumnCoverage` are computed across all contract rows (unavailable rows count as 0 coverage).
- `MeanColumnCoverageAvailable` and `MinColumnCoverageAvailable` summarize only rows whose source component is available.

`summary(out)` is supported through `summary()`. `plot(out)` is dispatched through `plot()` for class `mfrm_parity_report` (`type = "column_coverage", "table_coverage", "metric_status", "metric_by_table"`).

## Value

An object of class `mfrm_parity_report` with:

- `overall`: one-row compatibility-audit summary
- `column_summary`: coverage summary by table ID
- `column_audit`: row-level contract audit
- `missing_preview`: lowest-coverage rows
- `metric_summary`: one-row metric-check summary
- `metric_by_table`: metric-check summary by table ID
- `metric_audit`: row-level metric checks
- `settings`: branch/contract metadata

## Interpreting output

- `overall`: high-level compatibility-contract coverage and metric-check pass rates.
- `column_summary / column_audit`: where compatibility-schema mismatches occur.
- `metric_summary / metric_audit`: numerical consistency checks tied to the current contract.
- `missing_preview`: quickest path to unresolved compatibility gaps.

## Typical workflow

1. Run `facets_parity_report(fit, branch = "facets")`.
2. Inspect `summary(parity)` and `missing_preview`.
3. Patch upstream table builders, then rerun the compatibility audit.

**See Also**

[fit\\_mfrm\(\)](#), [diagnose\\_mfrm\(\)](#), [build\\_fixed\\_reports\(\)](#), [mfrmr\\_compatibility\\_layer](#)

**Examples**

```
toy <- load_mfrm_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
parity <- facets_parity_report(fit, diagnostics = diag, branch = "facets")
summary(parity)
p <- plot(parity, draw = FALSE)
```

---

facet\_quality\_dashboard

*Facet-quality dashboard for facet-level screening*

---

**Description**

Build a compact dashboard for one facet at a time, combining facet severity, misfit, central-tendency screening, and optional bias counts.

**Usage**

```
facet_quality_dashboard(
  fit,
  diagnostics = NULL,
  facet = NULL,
  bias_results = NULL,
  severity_warn = 1,
  misfit_warn = 1.5,
  central_tendency_max = 0.25,
  bias_count_warn = 1L,
  bias_abs_t_warn = 2,
  bias_abs_size_warn = 0.5,
  bias_p_max = 0.05
)
```

**Arguments**

<code>fit</code>	Output from <a href="#">fit_mfrm()</a> .
<code>diagnostics</code>	Optional output from <a href="#">diagnose_mfrm()</a> .
<code>facet</code>	Optional facet name. When NULL, the function tries to infer a rater-like facet and otherwise falls back to the first modeled facet.
<code>bias_results</code>	Optional output from <a href="#">estimate_bias()</a> or a named list of such outputs. Non-matching bundles are skipped quietly.

severity_warn	Absolute estimate cutoff used to flag severity outliers.
misfit_warn	Mean-square cutoff used to flag misfit. Values above this cutoff or below its reciprocal are flagged.
central_tendency_max	Absolute estimate cutoff used to flag central tendency. Levels near zero are marked.
bias_count_warn	Minimum flagged-bias row count required to flag a level.
bias_abs_t_warn	Absolute t cutoff used when deriving bias-row flags from a raw bias bundle.
bias_abs_size_warn	Absolute bias-size cutoff used when deriving bias-row flags from a raw bias bundle.
bias_p_max	Probability cutoff used when deriving bias-row flags from a raw bias bundle.

## Details

The dashboard screens individual facet elements across four complementary criteria:

- **Severity:** elements with  $|\text{Estimate}| > \text{severity\_warn}$  logits are flagged as unusually harsh or lenient.
- **Misfit:** elements with Infit or Outfit MnSq outside  $[1/\text{misfit\_warn}, \text{misfit\_warn}]$  (default 0.67–1.5) are flagged.
- **Central tendency:** elements with  $|\text{Estimate}| < \text{central\_tendency\_max}$  logits are flagged. Near-zero estimates may indicate a rater who avoids extreme categories, producing artificially narrow score ranges.
- **Bias:** elements involved in  $\geq \text{bias\_count\_warn}$  screen-positive interaction cells (from `estimate_bias()`) are flagged.

A **flag density** score counts how many of the four criteria each element triggers. Elements flagged on multiple criteria warrant priority review (e.g., rater retraining, data exclusion).

Default thresholds are designed for moderate-stakes rating contexts. Adjust for your application: stricter thresholds for high-stakes certification, more lenient for formative assessment.

## Value

An object of class `mfrm_facet_dashboard` (also inheriting from `mfrm_bundle` and `list`). The object summarizes one target facet: `overview` reports the facet-level screening totals, `summary` provides aggregate estimates and flag counts, `detail` contains one row per facet level with the computed screening indicators, `ranked` orders levels by review priority, `flagged` keeps only levels requiring follow-up, `bias_sources` records which bias-result bundles contributed to the counts, `settings` stores the resolved thresholds, and `notes` gives short interpretation messages about how to read the dashboard.

## Output

The returned object is a bundle-like list with class `mfrm_facet_dashboard` and components such as:

- overview: one-row structural overview
- summary: one-row screening summary
- detail: level-level detail table
- ranked: detail ordered by flag density / severity
- flagged: flagged levels only
- bias\_sources: per-bundle bias aggregation metadata
- settings: resolved threshold settings
- notes: short interpretation notes

## See Also

[diagnose\\_mfrm\(\)](#), [estimate\\_bias\(\)](#), [plot\\_qc\\_dashboard\(\)](#)

## Examples

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
dash <- facet_quality_dashboard(fit, diagnostics = diag)
summary(dash)
```

---

facet\_statistics\_report

*Build a facet statistics report (preferred alias)*

---

## Description

Build a facet statistics report (preferred alias)

## Usage

```
facet_statistics_report(  
  fit,  
  diagnostics = NULL,  
  metrics = c("Estimate", "Infit", "Outfit", "SE"),  
  ruler_width = 41,  
  distribution_basis = c("both", "sample", "population"),  
  se_mode = c("both", "model", "fit_adjusted")  
)
```

**Arguments**

<code>fit</code>	Output from <code>fit_mfrm()</code> .
<code>diagnostics</code>	Optional output from <code>diagnose_mfrm()</code> .
<code>metrics</code>	Numeric columns in <code>diagnostics\$measures</code> to summarize.
<code>ruler_width</code>	Width of the fixed-width ruler used for M/S/Q/X marks.
<code>distribution_basis</code>	Which distribution basis to keep in the appended precision summary: "both" (default), "sample", or "population".
<code>se_mode</code>	Which standard-error mode to keep in the appended precision summary: "both" (default), "model", or "fit_adjusted".

**Details**

`summary(out)` is supported through `summary()`. `plot(out)` is dispatched through `plot()` for class `mfrm_facet_statistics` (`type = "means", "sds", "ranges"`).

**Value**

A named list with facet-statistics components. Class: `mfrm_facet_statistics`.

**Interpreting output**

- facet-level means/SD/ranges of selected metrics (Estimate, fit indices, SE).
- fixed-width ruler rows (M/S/Q/X) for compact profile scanning.

**Typical workflow**

1. Run `facet_statistics_report(fit)`.
2. Inspect `summary/ranges` for anomalous facets.
3. Cross-check flagged facets with fit and chi-square diagnostics. The returned bundle now includes:
  - `precision_summary`: facet precision/separation indices by `DistributionBasis` and `SEMode`
  - `variability_tests`: fixed/random variability tests by facet
  - `se_modes`: compact list of available SE modes by facet

**See Also**

[diagnose\\_mfrm\(\)](#), [summary.mfrm\\_fit\(\)](#), [plot\\_facets\\_chisq\(\)](#), [mfrmr\\_reports\\_and\\_tables](#)

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
out <- facet_statistics_report(fit)
summary(out)
p_fs <- plot(out, draw = FALSE)
class(p_fs)
```

---

fair\_average\_table      *Build an adjusted-score reference table bundle*

---

### Description

Build an adjusted-score reference table bundle

### Usage

```
fair_average_table(
  fit,
  diagnostics = NULL,
  facets = NULL,
  totalscore = TRUE,
  umean = 0,
  uscale = 1,
  udecimals = 2,
  reference = c("both", "mean", "zero"),
  label_style = c("both", "native", "legacy"),
  omit_unobserved = FALSE,
  xtreme = 0
)
```

### Arguments

fit	Output from <code>fit_mfrm()</code> .
diagnostics	Optional output from <code>diagnose_mfrm()</code> .
facets	Optional subset of facets.
totalscore	Include all observations for score totals (TRUE) or apply legacy extreme-row exclusion (FALSE).
umean	Additive score-to-report origin shift.
uscale	Multiplicative score-to-report scale.
udecimals	Rounding digits used in formatted output.
reference	Which adjusted-score reference to keep in formatted outputs: "both" (default), "mean", or "zero".
label_style	Column-label style for formatted outputs: "both" (default), "native", or "legacy".
omit_unobserved	If TRUE, remove unobserved levels.
xtreme	Extreme-score adjustment amount.

### Details

This function wraps the package's adjusted-score calculations and returns both facet-wise and stacked tables. Historical display columns such as Fair(M) Average and Fair(Z) Average are retained for compatibility, and package-native aliases such as AdjustedAverage, StandardizedAdjustedAverage, ModelBasedSE, and FitAdjustedSE are appended to the formatted outputs.

**Value**

A named list with:

- `by_facet`: named list of formatted data.frames
- `stacked`: one stacked data.frame across facets
- `raw_by_facet`: unformatted internal tables
- `settings`: resolved options

**Interpreting output**

- `stacked`: cross-facet table for global comparison.
- `by_facet`: per-facet formatted tables for reporting.
- `raw_by_facet`: unformatted values for custom analyses/plots.
- `settings`: scoring-transformation and filtering options used.

Larger observed-vs-fair gaps can indicate systematic scoring tendencies by specific facet levels.

**Typical workflow**

1. Run `fair_average_table(fit, ...)`.
2. Inspect `summary(t12)` and `t12$stacked`.
3. Visualize with `plot_fair_average()`.

**Output columns**

The stacked data.frame contains:

**Facet** Facet name for this row.

**Level** Element label within the facet.

**Obsvd Average** Observed raw-score average.

**Fair(M) Average** Model-adjusted reference average on the reported score scale.

**Fair(Z) Average** Standardized adjusted reference average.

**ObservedAverage, AdjustedAverage, StandardizedAdjustedAverage** Package-native aliases for the three average columns above.

**Measure** Estimated logit measure for this level.

**SE** Compatibility alias for the model-based standard error.

**ModelBasedSE, FitAdjustedSE** Package-native aliases for Model S.E. and Real S.E..

**Infit MnSq, Outfit MnSq** Fit statistics for this level.

**See Also**

`diagnose_mfrm()`, `unexpected_response_table()`, `displacement_table()`

**Examples**

```

toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
t12 <- fair_average_table(fit, udecimals = 2)
t12_native <- fair_average_table(fit, reference = "mean", label_style = "native")
summary(t12)
p_t12 <- plot(t12, draw = FALSE)
class(p_t12)

```

---

fit\_mfrm

*Fit a many-facet Rasch model with a flexible number of facets*


---

**Description**

This is the package entry point. It wraps `mfrm_estimate()` and defaults to `method = "MML"`. Any number of facet columns can be supplied via `facets`.

**Usage**

```

fit_mfrm(
  data,
  person,
  facets,
  score,
  rating_min = NULL,
  rating_max = NULL,
  weight = NULL,
  keep_original = FALSE,
  model = c("RSM", "PCM"),
  method = c("MML", "JML", "JMLE"),
  step_facet = NULL,
  anchors = NULL,
  group_anchors = NULL,
  noncenter_facet = "Person",
  dummy_facets = NULL,
  positive_facets = NULL,
  anchor_policy = c("warn", "error", "silent"),
  min_common_anchors = 5L,
  min_obs_per_element = 30,
  min_obs_per_category = 10,
  quad_points = 15,
  maxit = 400,
  reltol = 1e-06
)

```

**Arguments**

data	A data.frame with one row per observation.
person	Column name for the person (character scalar).
facets	Character vector of facet column names.
score	Column name for observed category score.
rating_min	Optional minimum category value.
rating_max	Optional maximum category value.
weight	Optional weight column name.
keep_original	Keep original category values.
model	"RSM" or "PCM".
method	"MML" (default) or "JML" / "JMLE".
step_facet	Step facet for PCM.
anchors	Optional anchor table.
group_anchors	Optional group-anchor table.
noncenter_facet	One facet to leave non-centered.
dummy_facets	Facets to fix at zero.
positive_facets	Facets with positive orientation.
anchor_policy	How to handle anchor-audit issues: "warn" (default), "error", or "silent".
min_common_anchors	Minimum anchored levels per linking facet used in anchor-audit recommendations.
min_obs_per_element	Minimum weighted observations per facet level used in anchor-audit recommendations.
min_obs_per_category	Minimum weighted observations per score category used in anchor-audit recommendations.
quad_points	Quadrature points for MML.
maxit	Maximum optimizer iterations.
reltol	Optimization tolerance.

**Details**

Data must be in **long format** (one row per observed rating event).

**Value**

An object of class `mfrm_fit` (named list) with:

- summary: one-row model summary (LogLik, AIC, BIC, convergence)

- `facets$person`: person estimates (Estimate; plus SD for MML)
- `facets$others`: facet-level estimates for each facet
- `steps`: estimated threshold/step parameters
- `config`: resolved model configuration used for estimation (includes `config$anchor_audit`)
- `prep`: preprocessed data/level metadata
- `opt`: raw optimizer result from `stats::optim()`

## Model

`fit_mfrm()` estimates the many-facet Rasch model (Linacre, 1989). For a two-facet design (rater  $j$ , criterion  $i$ ) the model is:

$$\ln \frac{P(X_{nij} = k)}{P(X_{nij} = k - 1)} = \theta_n - \delta_j - \beta_i - \tau_k$$

where  $\theta_n$  is person ability,  $\delta_j$  rater severity,  $\beta_i$  criterion difficulty, and  $\tau_k$  the  $k$ -th Rasch-Andrich threshold. Any number of facets may be specified via the `facets` argument; each enters as an additive term in the linear predictor  $\eta$ .

With `model = "RSM"`, thresholds  $\tau_k$  are shared across all levels of all facets. With `model = "PCM"`, each level of `step_facet` receives its own threshold vector  $\tau_{i,k}$  on the package's shared observed score scale.

With `method = "MML"`, person parameters are integrated out using Gauss-Hermite quadrature and EAP estimates are computed post-hoc. With `method = "JML"`, all parameters are estimated jointly as fixed effects. See the "Estimation methods" section of [mfrmr-package](#) for details.

## Input requirements

Minimum required columns are:

- person identifier (`person`)
- one or more facet identifiers (`facets`)
- observed score (`score`)

Scores are treated as ordered categories. If your observed categories do not start at 0, set `rating_min/rating_max` explicitly to avoid unintended recoding assumptions.

Supported model/estimation combinations:

- `model = "RSM"` with `method = "MML"` or `"JML"/"JMLE"`
- `model = "PCM"` with a designated `step_facet` (defaults to first facet)

Anchor inputs are optional:

- `anchors` should contain facet/level/fixed-value information.
- `group_anchors` should contain facet/level/group/group-value information. Both are normalized internally, so column names can be flexible (`facet`, `level`, `anchor`, `group`, `groupvalue`, etc.).

Anchor audit behavior:

- `fit_mfrm()` runs an internal anchor audit.
- invalid rows are removed before estimation.
- duplicate rows keep the last occurrence for each key.
- `anchor_policy` controls whether detected issues are warned, treated as errors, or kept silent.

Facet sign orientation:

- facets listed in `positive_facets` are treated as +1
- all other facets are treated as -1 This affects interpretation of reported facet measures.

### Performance tips

For exploratory work, `method = "JML"` is usually faster than `method = "MML"`, but it may require a larger `maxit` to converge on larger datasets.

For MML runs, `quad_points` is the main accuracy/speed trade-off:

- `quad_points = 7` is a good lightweight default for quick iteration.
- `quad_points = 15` gives a more stable approximation for final reporting.

Downstream diagnostics can also be staged:

- use `diagnose_mfrm(fit, residual_pca = "none")` for a quick first pass
- add residual PCA only when you need exploratory residual-structure evidence

Downstream diagnostics report `ModelSE / RealSE` columns and related reliability indices. For MML, non-person facet `ModelSE` values are based on the observed information of the marginal log-likelihood and person rows use posterior SDs from EAP scoring. For JML, these quantities remain exploratory approximations and should not be treated as equally formal.

### Interpreting output

A typical first-pass read is:

1. `fit$summary` for convergence and global fit indicators.
2. `summary(fit)` for human-readable overviews.
3. `diagnose_mfrm(fit)` for element-level fit, approximate separation/reliability, and warning tables.

### Typical workflow

1. Fit the model with `fit_mfrm(...)`.
2. Validate convergence and scale structure with `summary(fit)`.
3. Run `diagnose_mfrm()` and proceed to reporting with `build_apa_outputs()`.

### See Also

[diagnose\\_mfrm\(\)](#), [estimate\\_bias\(\)](#), [build\\_apa\\_outputs\(\)](#), [mfrm\\_workflow\\_methods](#), [mfrm\\_reporting\\_and\\_apa](#)

**Examples**

```

toy <- load_mfrmr_data("example_core")

fit <- fit_mfrm(
  data = toy,
  person = "Person",
  facets = c("Rater", "Criterion"),
  score = "Score",
  method = "JML",
  model = "RSM",
  maxit = 25
)
fit$summary
s_fit <- summary(fit)
s_fit$overview[, c("Model", "Method", "Converged")]
p_fit <- plot(fit, draw = FALSE)
class(p_fit)

# MML is the default:
fit_mml <- fit_mfrm(
  data = toy,
  person = "Person",
  facets = c("Rater", "Criterion"),
  score = "Score",
  model = "RSM",
  quad_points = 7,
  maxit = 25
)
summary(fit_mml)

# Next steps after fitting:
diag_mml <- diagnose_mfrm(fit_mml, residual_pca = "none")
chk <- reporting_checklist(fit_mml, diagnostics = diag_mml)
head(chk$checklist[, c("Section", "Item", "DraftReady")])

```

---

interrater\_agreement\_table

*Build an inter-rater agreement report*


---

**Description**

Build an inter-rater agreement report

**Usage**

```

interrater_agreement_table(
  fit,
  diagnostics = NULL,

```

```

    rater_facet = NULL,
    context_facets = NULL,
    exact_warn = 0.5,
    corr_warn = 0.3,
    include_precision = TRUE,
    top_n = NULL
  )

```

### Arguments

<code>fit</code>	Output from <code>fit_mfrm()</code> .
<code>diagnostics</code>	Optional output from <code>diagnose_mfrm()</code> .
<code>rater_facet</code>	Name of the rater facet. If NULL, inferred from facet names.
<code>context_facets</code>	Optional context facets used to match observations for agreement. If NULL, all remaining facets (including Person) are used.
<code>exact_warn</code>	Warning threshold for exact agreement.
<code>corr_warn</code>	Warning threshold for pairwise correlation.
<code>include_precision</code>	If TRUE, append rater severity spread indices from the facet precision summary when available.
<code>top_n</code>	Optional maximum number of pair rows to keep.

### Details

This helper computes pairwise rater agreement on matched contexts and returns both a pair-level table and a one-row summary. The output is package-native and does not require knowledge of legacy report numbering.

### Value

A named list with:

- `summary`: one-row inter-rater summary
- `pairs`: pair-level agreement table
- `settings`: applied options and thresholds

### Interpreting output

- `summary`: overall agreement level, number/share of flagged pairs.
- `pairs`: pairwise exact agreement, correlation, and direction/size gaps.
- `settings`: applied facet matching and warning thresholds.

Pairs flagged by both low exact agreement and low correlation generally deserve highest calibration priority.

**Typical workflow**

1. Run with explicit `rater_facet` (and `context_facets` if needed).
2. Review `summary(ir)` and top flagged rows in `ir$pairs`.
3. Visualize with `plot_interrater_agreement()`.

**Output columns**

The `pairs` data.frame contains:

**Rater1, Rater2** Rater pair identifiers.

**N** Number of matched-context observations for this pair.

**Exact** Proportion of exact score agreements.

**ExpectedExact** Expected exact agreement under chance.

**Adjacent** Proportion of adjacent (+/- 1 category) agreements.

**MeanDiff** Signed mean score difference (Rater1 - Rater2).

**MAD** Mean absolute score difference.

**Corr** Pearson correlation between paired scores.

**Flag** Logical; TRUE when `Exact < exact_warn` or `Corr < corr_warn`.

**OpportunityCount, ExactCount, ExpectedExactCount, AdjacentCount** Raw counts behind the agreement proportions.

The `summary` data.frame contains:

**RaterFacet** Name of the rater facet analyzed.

**TotalPairs** Number of rater pairs evaluated.

**ExactAgreement** Mean exact agreement across all pairs.

**AgreementMinusExpected** Observed exact agreement minus expected exact agreement.

**MeanCorr** Mean pairwise correlation.

**FlaggedPairs, FlaggedShare** Count and proportion of flagged pairs.

**RaterSeparation, RaterReliability** Severity-spread indices for the rater facet, reported separately from agreement.

**See Also**

[diagnose\\_mfrm\(\)](#), [facets\\_chisq\\_table\(\)](#), [plot\\_interrater\\_agreement\(\)](#), [mfrmr\\_visual\\_diagnostics](#)

**Examples**

```
toy <- load_mfrm_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
ir <- interrater_agreement_table(fit, rater_facet = "Rater")
summary(ir)
p_ir <- plot(ir, draw = FALSE)
class(p_ir)
```

---

list_mfrmr_data	<i>List packaged simulation datasets</i>
-----------------	--

---

## Description

List packaged simulation datasets

## Usage

```
list_mfrmr_data()
```

## Details

Use this helper when you want to select packaged data programmatically (e.g., inside scripts, loops, or shiny/streamlit wrappers).

Typical pattern:

1. call `list_mfrmr_data()` to see available keys.
2. pass one key to `load_mfrmr_data()`.

## Value

Character vector of dataset keys accepted by `load_mfrmr_data()`.

## Interpreting output

Returned values are canonical dataset keys accepted by `load_mfrmr_data()`.

## Typical workflow

1. Capture keys in a script (`keys <- list_mfrmr_data()`).
2. Select one key by index or name.
3. Load data via `load_mfrmr_data()` and continue analysis.

## See Also

[load\\_mfrmr\\_data\(\)](#), [ej2021\\_data](#)

## Examples

```
keys <- list_mfrmr_data()
keys
d <- load_mfrmr_data(keys[1])
head(d)
```

---

load_mfrmr_data	<i>Load a packaged simulation dataset</i>
-----------------	---

---

### Description

Load a packaged simulation dataset

### Usage

```
load_mfrmr_data(  
  name = c("example_core", "example_bias", "study1", "study2", "combined",  
           "study1_intercal", "study2_intercal", "combined_intercal")  
)
```

### Arguments

name                    Dataset key. One of values from [list\\_mfrmr\\_data\(\)](#).

### Details

This helper is useful in scripts/functions where you want to choose a dataset by string key instead of calling `data()` manually.

All returned datasets include the core long-format columns Study, Person, Rater, Criterion, and Score. Some datasets, such as the packaged documentation examples, also include auxiliary variables like Group for DIF/bias demonstrations.

### Value

A `data.frame` in long format.

### Interpreting output

The return value is a plain long-format `data.frame`, ready for direct use in [fit\\_mfrm\(\)](#) without additional reshaping.

### Typical workflow

1. list valid names with [list\\_mfrmr\\_data\(\)](#).
2. load one dataset key with `load_mfrmr_data(name)`.
3. fit a model with [fit\\_mfrm\(\)](#) and inspect with `summary()` / `plot()`.

### See Also

[list\\_mfrmr\\_data\(\)](#), [ej2021\\_data](#)

## Examples

```
data("mfrmr_example_core", package = "mfrmr")
head(mfrmr_example_core)

d <- load_mfrmr_data("example_core")
fit <- fit_mfrm(
  data = d,
  person = "Person",
  facets = c("Rater", "Criterion"),
  score = "Score",
  method = "JML",
  maxit = 25
)
summary(fit)
```

---

make_anchor_table	<i>Build an anchor table from fitted estimates</i>
-------------------	--

---

## Description

Build an anchor table from fitted estimates

## Usage

```
make_anchor_table(fit, facets = NULL, include_person = FALSE, digits = 6)
```

## Arguments

fit	Output from <code>fit_mfrm()</code> .
facets	Optional subset of facets to include.
include_person	Include person estimates as anchors.
digits	Rounding digits for anchor values.

## Details

This function exports estimated facet parameters as an anchor table for use in subsequent calibrations. This is the standard approach for **linking** across administrations: a reference run establishes the measurement scale, and anchored re-analyses place new data on that same scale.

Anchor values should be exported from a well-fitting reference run with adequate sample size. If the reference model has convergence issues or large misfit, the exported anchors may propagate instability. Re-run `audit_mfrm_anchors()` on the receiving data to verify compatibility before estimation.

The `digits` parameter controls rounding precision. Use at least 4 digits for research applications; excessive rounding (e.g., 1 digit) can introduce avoidable calibration error.

**Value**

A data.frame with Facet, Level, and Anchor.

**Interpreting output**

- Facet: facet name to be anchored in later runs.
- Level: specific element/level name inside that facet.
- Anchor: fixed logit value (rounded by digits).

**Typical workflow**

1. Fit a reference run with `fit_mfrm()`.
2. Export anchors with `make_anchor_table(fit)`.
3. Pass selected rows back into `fit_mfrm(..., anchors = ...)`.

**See Also**

`fit_mfrm()`, `audit_mfrm_anchors()`

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
anchors_tbl <- make_anchor_table(fit)
head(anchors_tbl)
summary(anchors_tbl$Anchor)
```

---

measurable\_summary\_table

*Build a measurable-data summary*

---

**Description**

Build a measurable-data summary

**Usage**

```
measurable_summary_table(fit, diagnostics = NULL)
```

**Arguments**

`fit` Output from `fit_mfrm()`.

`diagnostics` Optional output from `diagnose_mfrm()`.

**Details**

This helper consolidates measurable-data diagnostics into a dedicated report bundle: run-level summary, facet coverage, category usage, and subset (connected-component) information.

`summary(t5)` is supported through `summary()`. `plot(t5)` is dispatched through `plot()` for class `mfrm_measurable` (type = "facet\_coverage", "category\_counts", "subset\_observations").

**Value**

A named list with:

- `summary`: one-row measurable-data summary
- `facet_coverage`: per-facet coverage summary
- `category_stats`: category-level usage/fit summary
- `subsets`: subset summary table (when available)

**Interpreting output**

- `summary`: overall measurable design status.
- `facet_coverage`: spread/precision by facet.
- `category_stats`: category usage and fit context.
- `subsets`: connectivity diagnostics (fragmented subsets reduce comparability).

**Typical workflow**

1. Run `measurable_summary_table(fit)`.
2. Check `summary(t5)` for subset/connectivity warnings.
3. Use `plot(t5, ...)` to inspect facet/category/subset views.

**Further guidance**

For a plot-selection guide and a longer walkthrough, see [mfrmr\\_visual\\_diagnostics](#) and `vignette("mfrmr-visual-diagnosis", package = "mfrmr")`.

**Output columns**

The `summary` data.frame (one row) contains:

**Observations, TotalWeight** Total observations and summed weight.

**Persons, Facets, Categories** Design dimensions.

**ConnectedSubsets** Number of connected subsets.

**LargestSubsetObs, LargestSubsetPct** Largest subset coverage.

The `facet_coverage` data.frame contains:

**Facet** Facet name.

**Levels** Number of estimated levels.

**MeanSE** Mean standard error across levels.

**MeanInfit, MeanOutfit** Mean fit statistics across levels.

**MinEstimate, MaxEstimate** Measure range for this facet.

The `category_stats` data.frame contains:

**Category** Score category value.

**Count, Percent** Observed count and percentage.

**Infit, Outfit, InfitZSTD, OutfitZSTD** Category-level fit.

**ExpectedCount, DiffCount, LowCount** Expected-observed comparison and low-count flag.

### See Also

[diagnose\\_mfrmr\(\)](#), [rating\\_scale\\_table\(\)](#), [describe\\_mfrmr\\_data\(\)](#), [mfrmr\\_visual\\_diagnostics](#)

### Examples

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrmr(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
t5 <- measurable_summary_table(fit)
summary(t5)
p_t5 <- plot(t5, draw = FALSE)
class(p_t5)
```

---

mfrmr\_compatibility\_layer

*mfrmr Compatibility Layer Map*

---

### Description

Guide to the legacy-compatible wrappers and text/file exports in `mfrmr`. Use this page when you need continuity with older compatibility-oriented workflows, fixed-width reports, or graph/score file style outputs.

### When to use this layer

- You are reproducing an older workflow that expects one-shot wrappers.
- You need fixed-width text blocks for console, logs, or archival handoff.
- You need graphfile or scorefile style outputs for downstream legacy tools.
- You are checking column/metric parity against a compatibility contract.

### When not to use this layer

- For standard estimation, use `fit_mfrmr()` plus `diagnose_mfrmr()`.
- For report bundles, use `mfrmr_reports_and_tables`.
- For manuscript text, use `build_apa_outputs()` and `reporting_checklist()`.
- For visual follow-up, use `mfrmr_visual_diagnostics`.

### Compatibility map

[run\\_mfrmr\\_facets\(\)](#) One-shot legacy-compatible wrapper that fits, diagnoses, and returns key tables in one object.

[mfrmrRFacets\(\)](#) Alias for [run\\_mfrmr\\_facets\(\)](#) kept for continuity.

[build\\_fixed\\_reports\(\)](#) Fixed-width interaction and pairwise text blocks. Best when a text-only compatibility artifact is required.

[facets\\_output\\_file\\_bundle\(\)](#) Graphfile/scorefile style CSV and fixed-width exports for legacy pipelines.

[facets\\_parity\\_report\(\)](#) Column and metric contract audit against the compatibility specification. Use only when explicit parity checking is part of the task.

### Preferred replacements

- Instead of [run\\_mfrmr\\_facets\(\)](#), prefer: [fit\\_mfrmr\(\)](#) -> [diagnose\\_mfrmr\(\)](#) -> [reporting\\_checklist\(\)](#).
- Instead of [build\\_fixed\\_reports\(\)](#), prefer: [bias\\_interaction\\_report\(\)](#) -> [build\\_apa\\_outputs\(\)](#).
- Instead of [facets\\_output\\_file\\_bundle\(\)](#), prefer: [category\\_curves\\_report\(\)](#) or [category\\_structure\\_report\(\)](#) plus [export\\_mfrmr\\_bundle\(\)](#).
- Instead of [facets\\_parity\\_report\(\)](#) for routine QA, prefer: [reference\\_case\\_audit\(\)](#) for package-native completeness auditing or [reference\\_case\\_benchmark\(\)](#) for internal benchmark cases.

### Practical migration rules

- Keep compatibility wrappers only where a downstream consumer truly needs the old layout or fixed-width format.
- For new scripts, start from package-native bundles and add compatibility outputs only at the export boundary.
- Treat compatibility outputs as presentation contracts, not as the primary analysis objects.

### Typical workflow

- Legacy handoff: [run\\_mfrmr\\_facets\(\)](#) -> [build\\_fixed\\_reports\(\)](#) -> [facets\\_output\\_file\\_bundle\(\)](#).
- Mixed workflow: [fit\\_mfrmr\(\)](#) -> [diagnose\\_mfrmr\(\)](#) -> [build\\_apa\\_outputs\(\)](#) -> compatibility export only if required.
- Contract audit: [fit\\_mfrmr\(\)](#) -> [diagnose\\_mfrmr\(\)](#) -> [facets\\_parity\\_report\(\)](#).

### Companion guides

- For standard reports/tables, see [mfrmr\\_reports\\_and\\_tables](#).
- For manuscript-draft reporting, see [mfrmr\\_reporting\\_and\\_apa](#).
- For visual diagnostics, see [mfrmr\\_visual\\_diagnostics](#).
- For linking and DFF workflows, see [mfrmr\\_linking\\_and\\_dff](#).
- For end-to-end routes, see [mfrmr\\_workflow\\_methods](#).

## Examples

```
toy <- load_mfrmr_data("example_core")
toy_small <- toy[toy$Person %in% unique(toy$Person)[1:12], , drop = FALSE]

run <- run_mfrmr_facets(
  data = toy_small,
  person = "Person",
  facets = c("Rater", "Criterion"),
  score = "Score",
  maxit = 10
)
summary(run)

fixed <- build_fixed_reports(
  estimate_bias(
    run$fit,
    run$diagnostics,
    facet_a = "Rater",
    facet_b = "Criterion",
    max_iter = 1
  ),
  branch = "original"
)
names(fixed)
```

---

mfrmr\_example\_data      *Purpose-built example datasets for package help pages*

---

## Description

Compact synthetic many-facet datasets designed for documentation examples. Both datasets are large enough to avoid tiny-sample toy behavior while remaining fast in R CMD check examples.

## Format

A data.frame with 6 columns:

**Study** Example dataset label ("ExampleCore" or "ExampleBias").

**Person** Person/respondent identifier.

**Rater** Rater identifier.

**Criterion** Criterion facet label.

**Score** Observed category score on a four-category scale (1–4).

**Group** Balanced grouping variable used in DFF/DIF examples ("A" / "B").

## Details

Available data objects:

- `mfrmr_example_core`
- `mfrmr_example_bias`

`mfrmr_example_core` is generated from a single latent trait plus rater and criterion main effects, making it suitable for general fitting, plotting, and reporting examples.

`mfrmr_example_bias` starts from the same basic design but adds:

- a known Group x Criterion effect (Group B is advantaged on Language)
- a known Rater x Criterion interaction (R04 x Accuracy)

This lets differential-functioning and bias-analysis help pages demonstrate non-null findings.

## Data dimensions

Dataset	Rows	Persons	Raters	Criteria	Groups
<code>example_core</code>	768	48	4	4	2
<code>example_bias</code>	384	48	4	4	2

## Suggested usage

- Use `mfrmr_example_core` for fitting, diagnostics, design-weighted precision curves, and generic plots/reports.
- Use `mfrmr_example_bias` for `analyze_dff()`, `analyze_dif()`, `dif_interaction_table()`, `plot_dif_heatmap()`, and `estimate_bias()`.

Both objects can be loaded either with `load_mfrmr_data()` or directly via `data("mfrmr_example_core", package = "mfrmr") / data("mfrmr_example_bias", package = "mfrmr")`.

## Source

Synthetic documentation data generated from rating-scale Rasch facet designs with fixed seeds in `data-raw/make-example-data.R`.

## Examples

```
data("mfrmr_example_core", package = "mfrmr")
table(mfrmr_example_core$Score)
table(mfrmr_example_core$Group)
```

## Description

Package-native guide to checking connectedness, building anchor-based links, monitoring drift, and screening differential facet functioning (DFF) in `mfrmr`.

## Start with the linking question

- "Is the design connected enough to support a common scale?" Use `subset_connectivity_report()` and `plot(..., type = "design_matrix")`.
- "Which elements can I export as anchors from an existing fit?" Use `make_anchor_table()` and `audit_mfrm_anchors()`.
- "How do I anchor a new administration to a baseline?" Use `anchor_to_baseline()`.
- "Have common elements drifted across separately fitted waves?" Use `detect_anchor_drift()` and `plot_anchor_drift()`.
- "Do specific facet levels function differently across groups?" Use `analyze_dff()` and `plot_dif_heatmap()`.

## Recommended linking route

1. Fit with `fit_mfrm()` and diagnose with `diagnose_mfrm()`.
2. Check connectedness with `subset_connectivity_report()`.
3. Build or audit anchors with `make_anchor_table()` and `audit_mfrm_anchors()`.
4. Use `anchor_to_baseline()` when you need to place raw new data onto a baseline scale.
5. Use `build_equating_chain()` only as a screened linking aid across already fitted waves.
6. Use `detect_anchor_drift()` for stability monitoring on separately fitted waves.
7. Run `analyze_dff()` only after checking connectivity and common-scale evidence.

## Which helper answers which task

`subset_connectivity_report()` Summarizes connected subsets, bottleneck facets, and design-matrix coverage.

`make_anchor_table()` Extracts reusable anchor candidates from a fit.

`anchor_to_baseline()` Anchors new raw data to a baseline fit and returns anchored diagnostics plus a consistency check against the baseline scale.

`detect_anchor_drift()` Compares fitted waves directly to flag unstable anchor elements.

`build_equating_chain()` Accumulates screened pairwise links across a series of administrations or forms.

`analyze_dff()` Screens differential facet functioning with residual or refit methods, using screening-only language unless linking and precision support stronger interpretation.

### Practical linking rules

- Check connectedness before interpreting subgroup or wave differences.
- Use DFF outputs as screening results when common-scale linking is weak.
- Treat drift flags as prompts for review, not automatic evidence that an anchor must be removed.
- Treat `LinkSupportAdequate = FALSE` as a weak-link warning: at least one linking facet retained fewer than 5 common elements after screening.
- Rebuild anchors from a defensible baseline rather than chaining unstable links by hand.

### Typical workflow

- Cross-sectional linkage review: `fit_mfrmr() -> diagnose_mfrmr() -> subset_connectivity_report() -> plot(..., type = "design_matrix")`.
- Baseline placement review: `make_anchor_table() -> anchor_to_baseline() -> diagnose_mfrmr()`.
- Multi-wave drift review: fit each wave separately -> `detect_anchor_drift() -> plot_anchor_drift()`.
- Group comparison route: `subset_connectivity_report() -> analyze_dff() -> dif_report() -> plot_dif_heatmap()`.

### Companion guides

- For visual follow-up, see [mfrmr\\_visual\\_diagnostics](#).
- For report/table selection, see [mfrmr\\_reports\\_and\\_tables](#).
- For end-to-end routes, see [mfrmr\\_workflow\\_methods](#).
- For a longer walkthrough, see `vignette("mfrmr-linking-and-dff", package = "mfrmr")`.

### Examples

```
toy <- load_mfrmr_data("example_bias")
fit <- fit_mfrmr(
  toy,
  person = "Person",
  facets = c("Rater", "Criterion"),
  score = "Score",
  method = "JML",
  maxit = 10
)
diag <- diagnose_mfrmr(fit, residual_pca = "none")

subsets <- subset_connectivity_report(fit, diagnostics = diag)
subsets$summary[, c("Subset", "Observations", "ObservationPercent")]

dff <- analyze_dff(fit, diag, facet = "Rater", group = "Group", data = toy)
head(dff$dif_table[, c("Level", "Group1", "Group2", "Classification")])
```

---

 mfrmr\_reporting\_and\_apa

*mfrmr Reporting and APA Guide*


---

## Description

Package-native guide to moving from fitted model objects to manuscript-draft text, tables, notes, and revision checklists in `mfrmr`.

## Start with the reporting question

- "Which parts of this run are draft-complete, and with what caveats?" Use `reporting_checklist()`.
- "How should I phrase the model, fit, and precision sections?" Use `build_apa_outputs()`.
- "Which tables should I hand off to a manuscript or appendix?" Use `apa_table()` and `facet_statistics_report()`.
- "How do I explain model-based vs exploratory precision?" Use `precision_audit_report()` and `summary(diagnose_mfrm(...))`.
- "Which caveats need to appear in the write-up?" Use `reporting_checklist()` first, then `build_apa_outputs()`.

## Recommended reporting route

1. Fit with `fit_mfrm()`.
2. Build diagnostics with `diagnose_mfrm()`.
3. Review precision strength with `precision_audit_report()` when inferential language matters.
4. Run `reporting_checklist()` to identify missing sections, caveats, and next actions.
5. Create manuscript-draft prose and metadata with `build_apa_outputs()`.
6. Convert specific components to handoff tables with `apa_table()`.

## Which helper answers which task

`reporting_checklist()` Turns current analysis objects into a prioritized revision guide with `DraftReady`, `Priority`, and `NextAction`. `DraftReady` means "ready to draft with the documented caveats"; `ReadyForAPA` is retained as a backward-compatible alias, and neither field means "formal inference is automatically justified".

`build_apa_outputs()` Builds shared-contract prose, table notes, captions, and a section map from the current fit and diagnostics.

`apa_table()` Produces reproducible base-R tables with APA-oriented labels, notes, and captions.

`precision_audit_report()` Summarizes whether precision claims are model-based, hybrid, or exploratory.

`facet_statistics_report()` Provides facet-level summaries that often feed result tables and appendix material.

`build_visual_summaries()` Prepares publication-oriented figure payloads that can be cited from the report text.

### Practical reporting rules

- Treat `reporting_checklist()` as the gap finder and `build_apa_outputs()` as the writing engine.
- Phrase formal inferential claims only when the precision tier is model-based.
- Keep bias and differential-functioning outputs in screening language unless the current precision layer and linking evidence justify stronger claims.
- Treat `DraftReady` (and the legacy alias `ReadyForAPA`) as a drafting-readiness flag, not as a substitute for methodological review.
- Rebuild APA outputs after major model changes instead of editing old text by hand.

### Typical workflow

- Manuscript-first route: `fit_mfrm()` -> `diagnose_mfrm()` -> `reporting_checklist()` -> `build_apa_outputs()` -> `apa_table()`.
- Appendix-first route: `facet_statistics_report()` -> `apa_table()` -> `build_visual_summaries()` -> `build_apa_outputs()`.
- Precision-sensitive route: `diagnose_mfrm()` -> `precision_audit_report()` -> `reporting_checklist()` -> `build_apa_outputs()`.

### Companion guides

- For report/table selection, see [mfrmr\\_reports\\_and\\_tables](#).
- For end-to-end analysis routes, see [mfrmr\\_workflow\\_methods](#).
- For visual follow-up, see [mfrmr\\_visual\\_diagnostics](#).
- For a longer walkthrough, see `vignette("mfrmr-reporting-and-apa", package = "mfrmr")`.

### Examples

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(
  toy,
  person = "Person",
  facets = c("Rater", "Criterion"),
  score = "Score",
  method = "JML",
  maxit = 25
)
diag <- diagnose_mfrm(fit, residual_pca = "none")

checklist <- reporting_checklist(fit, diagnostics = diag)
head(checklist$checklist[, c("Section", "Item", "DraftReady", "NextAction")])

apa <- build_apa_outputs(fit, diagnostics = diag)
names(apa$section_map)

tbl <- apa_table(fit, which = "summary")
tbl$caption
```

---

mfrmr\_reports\_and\_tables

*mfrmr Reports and Tables Map*

---

## Description

Quick guide to choosing the right report or table helper in mfrmr. Use this page when you know the reporting question but have not yet decided which bundle, table, or reporting helper to call.

## Start with the question

- "How should I document the model setup and run settings?" Use [specifications\\_report\(\)](#).
- "Was data filtered, dropped, or mapped in unexpected ways?" Use [data\\_quality\\_report\(\)](#) and [describe\\_mfrm\\_data\(\)](#).
- "Did estimation converge cleanly and how formal is the precision layer?" Use [estimation\\_iteration\\_report\(\)](#) and [precision\\_audit\\_report\(\)](#).
- "Which facets are measurable, variable, or weakly separated?" Use [facet\\_statistics\\_report\(\)](#), [measurable\\_summary\\_table\(\)](#), and [facets\\_chisq\\_table\(\)](#).
- "Are score categories functioning in a usable sequence?" Use [rating\\_scale\\_table\(\)](#), [category\\_structure\\_report\(\)](#) and [category\\_curves\\_report\(\)](#).
- "Is the design linked well enough across subsets, forms, or waves?" Use [subset\\_connectivity\\_report\(\)](#) and [plot\\_anchor\\_drift\(\)](#).
- "What should go into the manuscript text and tables?" Use [reporting\\_checklist\(\)](#) and [build\\_apa\\_outputs\(\)](#).

## Recommended report route

1. Start with [specifications\\_report\(\)](#) and [data\\_quality\\_report\(\)](#) to document the run and confirm usable data.
2. Continue with [estimation\\_iteration\\_report\(\)](#) and [precision\\_audit\\_report\(\)](#) to judge convergence and inferential strength.
3. Use [facet\\_statistics\\_report\(\)](#) and [subset\\_connectivity\\_report\(\)](#) to describe spread, linkage, and measurability.
4. Add [rating\\_scale\\_table\(\)](#), [category\\_structure\\_report\(\)](#), and [category\\_curves\\_report\(\)](#) to document scale functioning.
5. Finish with [reporting\\_checklist\(\)](#) and [build\\_apa\\_outputs\(\)](#) for manuscript-oriented output.

## Which output answers which question

[specifications\\_report\(\)](#) Documents model type, estimation method, anchors, and core run settings. Best for method sections and audit trails.

[data\\_quality\\_report\(\)](#) Summarizes retained and dropped rows, missingness, and unknown elements. Best for data cleaning narratives.

- `estimation_iteration_report()` Shows replayed convergence trajectories. Best for diagnosing slow or unstable estimation.
- `precision_audit_report()` Summarizes whether SE, CI, and reliability indices are model-based, hybrid, or exploratory. Best for deciding how strongly to phrase inferential claims.
- `facet_statistics_report()` Bundles facet summaries, precision summaries, and variability tests. Best for facet-level reporting.
- `subset_connectivity_report()` Summarizes disconnected subsets and coverage bottlenecks. Best for linking and anchor strategy review.
- `rating_scale_table()` Gives category counts, average measures, and threshold diagnostics. Best for first-pass category evaluation.
- `category_structure_report()` Adds transition points and compact category warnings. Best for category-order interpretation.
- `category_curves_report()` Returns category-probability curve coordinates and summaries. Best for downstream graphics and report drafts.
- `reporting_checklist()` Turns analysis status into an action list with priorities and next steps. Best for closing reporting gaps.
- `build_apa_outputs()` Creates manuscript-draft text, notes, captions, and section maps from a shared reporting contract.

### Practical interpretation rules

- Use bundle summaries first, then drill down into component tables.
- Treat `precision_audit_report()` as the gatekeeper for formal inference.
- Treat category and bias outputs as complementary layers rather than substitutes for overall fit review.
- Use `reporting_checklist()` before `build_apa_outputs()` when a report still needs missing diagnostics or clearer caveats.

### Typical workflow

- Run documentation: `fit_mfrmr()` -> `specifications_report()` -> `data_quality_report()`.
- Precision and facet review: `diagnose_mfrmr()` -> `precision_audit_report()` -> `facet_statistics_report()`.
- Scale review: `rating_scale_table()` -> `category_structure_report()` -> `category_curves_report()`.
- Manuscript handoff: `reporting_checklist()` -> `build_apa_outputs()`.

### Companion guides

- For visual follow-up, see `mfrmr_visual_diagnostics`.
- For one-shot analysis routes, see `mfrmr_workflow_methods`.
- For manuscript assembly, see `mfrmr_reporting_and_apa`.
- For linking and DFF review, see `mfrmr_linking_and_dff`.
- For legacy-compatible wrappers and exports, see `mfrmr_compatibility_layer`.

**Examples**

```

toy <- load_mfrmr_data("example_core")
toy_small <- toy[toy$Person %in% unique(toy$Person)[1:12], , drop = FALSE]
fit <- fit_mfrmr(
  toy_small,
  person = "Person",
  facets = c("Rater", "Criterion"),
  score = "Score",
  method = "JML",
  maxit = 10
)
diag <- diagnose_mfrmr(fit, residual_pca = "none")

spec <- specifications_report(fit)
summary(spec)

prec <- precision_audit_report(fit, diagnostics = diag)
summary(prec)

checklist <- reporting_checklist(fit, diagnostics = diag)
names(checklist)

apa <- build_apa_outputs(fit, diagnostics = diag)
names(apa$section_map)

```

---

mfrmr\_visual\_diagnostics

*mfrmr Visual Diagnostics Map*

---

**Description**

Quick guide to choosing the right base-R diagnostic plot in mfrmr. Use this page when you know the analysis question but do not yet know which plotting helper or plot() method to call.

**Start with the question**

- "Do persons and facet levels overlap on the same logit scale?" Use plot(fit, type = "wright") or plot\_wright\_unified().
- "Where do score categories transition across theta?" Use plot(fit, type = "pathway") and plot(fit, type = "ccc").
- "Is the design linked well enough across subsets or administrations?" Use plot(subset\_connectivity\_report(...), type = "design\_matrix") and plot\_anchor\_drift().
- "Which responses or levels look locally problematic?" Use plot\_unexpected() and plot\_displacement().
- "Do raters agree and do facets separate meaningfully?" Use plot\_interrater\_agreement() and plot\_facets\_chisq().
- "Is there notable residual structure after the main Rasch dimension?" Use plot\_residual\_pca().

- "Which interaction cells or facet levels drive bias screening results?" Use `plot_bias_interaction()`.
- "I need one compact triage screen first." Use `plot_qc_dashboard()`.

### Recommended visual route

1. Start with `plot_qc_dashboard()` for one-page triage.
2. Move to `plot_unexpected()`, `plot_displacement()`, and `plot_interrater_agreement()` for flagged local issues.
3. Use `plot(fit, type = "wright")`, `plot(fit, type = "pathway")`, and `plot_residual_pca()` for structural interpretation.
4. Use `plot(..., draw = FALSE)` when you want reusable plotting payloads instead of immediate graphics.
5. Use `preset = "publication"` when you want the package's cleaner manuscript-oriented styling.

### Which plot answers which question

`plot(fit, type = "wright")` Shared logit map of persons, facet levels, and step thresholds. Best for targeting and spread.

`plot(fit, type = "pathway")` Expected score by theta, with dominant-category strips. Best for scale progression.

`plot(fit, type = "ccc")` Category probability curves. Best for checking whether categories peak in sequence.

`plot_unexpected()` Observation-level surprises. Best for case review and local misfit triage.

`plot_displacement()` Level-wise anchor movement. Best for anchor robustness and residual calibration tension.

`plot_interrater_agreement()` Exact agreement, expected agreement, pairwise correlation, and agreement gaps. Best for rater consistency.

`plot_facets_chisq()` Facet variability and chi-square summaries. Best for checking whether a facet contributes meaningful spread.

`plot_residual_pca()` Residual structure after the Rasch dimension is removed. Best for exploratory residual-structure review, not as a standalone unidimensionality test.

`plot_bias_interaction()` Interaction-bias screening views for cells and facet profiles. Best for systematic departure from the additive main-effects model.

`plot_anchor_drift()` Anchor drift and screened linking-chain visuals. Best for multi-form or multi-wave linking review after checking retained common-element support.

### Practical interpretation rules

- Wright map: look for gaps between person density and facet/step locations; large gaps indicate weaker targeting.
- Pathway / CCC: look for monotone progression and clear category dominance bands; flat or overlapping curves suggest weak category separation.
- Unexpected / displacement: use as screening tools, not final evidence by themselves.

- Inter-rater agreement and facet variability address different questions: agreement concerns scoring consistency, whereas variability concerns whether facet elements are statistically distinguishable.
- Residual PCA and bias plots should be interpreted as follow-up layers after the main fit screen, not as first-pass diagnostics.

### Typical workflow

- Quick screening: `fit_mfrmr()` -> `diagnose_mfrmr()` -> `plot_qc_dashboard()`.
- Scale and targeting review: `plot(fit, type = "wright")` -> `plot(fit, type = "pathway")` -> `plot(fit, type = "ccc")`.
- Linking review: `subset_connectivity_report()` -> `plot(..., type = "design_matrix")` -> `plot_anchor_drift()`.
- Interaction review: `estimate_bias()` -> `plot_bias_interaction()` -> `reporting_checklist()`.

### Companion vignette

For a longer, plot-first walkthrough, run `vignette("mfrmr-visual-diagnostics", package = "mfrmr")`.

### See Also

[mfrmr\\_workflow\\_methods](#), [mfrmr\\_reports\\_and\\_tables](#), [mfrmr\\_reporting\\_and\\_apa](#), [mfrmr\\_linking\\_and\\_dff](#), [plot.mfrmr\\_fit\(\)](#), [plot\\_qc\\_dashboard\(\)](#), [plot\\_unexpected\(\)](#), [plot\\_displacement\(\)](#), [plot\\_interrater\\_agreement](#), [plot\\_facets\\_chisq\(\)](#), [plot\\_residual\\_pca\(\)](#), [plot\\_bias\\_interaction\(\)](#), [plot\\_anchor\\_drift\(\)](#)

### Examples

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrmr(
  toy,
  person = "Person",
  facets = c("Rater", "Criterion"),
  score = "Score",
  method = "JML",
  maxit = 25
)
diag <- diagnose_mfrmr(fit, residual_pca = "none")

qc <- plot_qc_dashboard(fit, diagnostics = diag, draw = FALSE, preset = "publication")
names(qc$data)

wright <- plot(fit, type = "wright", draw = FALSE, preset = "publication")
wright$data$preset

pca <- analyze_residual_pca(diag, mode = "overall")
scree <- plot_residual_pca(pca, plot_type = "scree", draw = FALSE, preset = "publication")
scree$data$preset
```

---

 mfrmr\_workflow\_methods

*mfrmr Workflow and Method Map*


---

## Description

Quick reference for end-to-end mfrmr analysis and for checking which output objects support `summary()` and `plot()`.

## Typical workflow

1. Fit a model with `fit_mfrmr()`. For final reporting, prefer `method = "MML"` unless you explicitly want a fast exploratory JML pass.
2. (Optional) Use `run_mfrmr_facets()` or `mfrmrRFacets()` for a legacy-compatible one-shot workflow wrapper.
3. Build diagnostics with `diagnose_mfrmr()`.
4. (Optional) Estimate interaction bias with `estimate_bias()`.
5. Generate reporting bundles: `apa_table()`, `build_fixed_reports()`, `build_visual_summaries()`.
6. (Optional) Audit report completeness with `reference_case_audit()`. Use `facets_parity_report()` only when you explicitly need the compatibility layer.
7. (Optional) Benchmark packaged reference cases with `reference_case_benchmark()` when you want an internal package-native benchmark/audit run.
8. (Optional) For design planning or future scoring, move to the simulation/prediction layer: `build_mfrmr_sim_spec()` / `extract_mfrmr_sim_spec()` -> `evaluate_mfrmr_design()` / `predict_mfrmr_population()` -> `predict_mfrmr_units()` / `sample_mfrmr_plausible_values()`. Fixed-calibration unit scoring currently requires an MML fit, and prediction export requires actual prediction objects in addition to `include = "predictions"`.
9. Use `summary()` for compact text checks and `plot()` (or dedicated plot helpers) for base-R visual diagnostics.

## Three practical routes

- Quick first pass: `fit_mfrmr()` -> `diagnose_mfrmr()` -> `plot_qc_dashboard()`.
- Linking and coverage review: `subset_connectivity_report()` -> `plot(..., type = "design_matrix")` -> `plot_wright_unified()`.
- Manuscript prep: `reporting_checklist()` -> `build_apa_outputs()` -> `apa_table()`.
- Design planning and forecasting: `build_mfrmr_sim_spec()` or `extract_mfrmr_sim_spec()` -> `evaluate_mfrmr_design()` -> `predict_mfrmr_population()` -> `predict_mfrmr_units()` or `sample_mfrmr_plausible_values()` from an MML fit -> `export_mfrmr_bundle(population_prediction = ..., unit_prediction = ..., plausible_values = ..., include = "predictions", ...)`.

## Interpreting output

This help page is a map, not an estimator:

- use it to decide function order,
- confirm which objects have `summary()/plot()` defaults,
- and identify when dedicated helper functions are needed.

## Objects with default `summary()` and `plot()` routes

- `mfrmr_fit`: `summary(fit)` and `plot(fit, ...)`.
- `mfrmr_diagnostics`: `summary(diag)`; plotting via dedicated helpers such as `plot_unexpected()`, `plot_displacement()`, `plot_qc_dashboard()`.
- `mfrmr_bias`: `summary(bias)` and `plot_bias_interaction()`.
- `mfrmr_data_description`: `summary(ds)` and `plot(ds, ...)`.
- `mfrmr_anchor_audit`: `summary(aud)` and `plot(aud, ...)`.
- `mfrmr_facets_run`: `summary(run)` and `plot(run, type = c("fit", "qc"), ...)`.
- `apa_table`: `summary(tbl)` and `plot(tbl, ...)`.
- `mfrmr_apa_outputs`: `summary(apa)` for compact diagnostics of report text.
- `mfrmr_threshold_profiles`: `summary(profiles)` for preset threshold grids.
- `mfrmr_population_prediction`: `summary(pred)` for design-level forecast tables.
- `mfrmr_unit_prediction`: `summary(pred)` for fixed-calibration unit-level posterior summaries.
- `mfrmr_plausible_values`: `summary(pv)` for draw-level uncertainty summaries.
- `mfrmr_bundle` families: `summary()` and class-aware `plot(bundle, ...)`. Key bundle classes now also use class-aware `summary(bundle)`: `mfrmr_unexpected`, `mfrmr_fair_average`, `mfrmr_displacement`, `mfrmr_interrater`, `mfrmr_facets_chisq`, `mfrmr_bias_interaction`, `mfrmr_rating_scale`, `mfrmr_category_structure`, `mfrmr_category_curves`, `mfrmr_measurable`, `mfrmr_unexpected_after_bias`, `mfrmr_output_bundle`, `mfrmr_residual_pca`, `mfrmr_specifications`, `mfrmr_data_quality`, `mfrmr_iteration_report`, `mfrmr_subset_connectivity`, `mfrmr_facet_statistics`, `mfrmr_parity_report`, `mfrmr_reference_audit`, `mfrmr_reference_benchmark`.

## `plot.mfrmr_bundle()` coverage

Default dispatch now covers:

- `mfrmr_unexpected`, `mfrmr_fair_average`, `mfrmr_displacement`
- `mfrmr_interrater`, `mfrmr_facets_chisq`, `mfrmr_bias_interaction`
- `mfrmr_bias_count`, `mfrmr_fixed_reports`, `mfrmr_visual_summaries`
- `mfrmr_category_structure`, `mfrmr_category_curves`, `mfrmr_rating_scale`
- `mfrmr_measurable`, `mfrmr_unexpected_after_bias`, `mfrmr_output_bundle`
- `mfrmr_residual_pca`, `mfrmr_specifications`, `mfrmr_data_quality`
- `mfrmr_iteration_report`, `mfrmr_subset_connectivity`, `mfrmr_facet_statistics`
- `mfrmr_parity_report`, `mfrmr_reference_audit`, `mfrmr_reference_benchmark`

For unknown bundle classes, use dedicated plotting helpers or custom base-R plots from component tables.

**See Also**

[fit\\_mfrm\(\)](#), [run\\_mfrm\\_facets\(\)](#), [mfrmRFacets\(\)](#), [diagnose\\_mfrm\(\)](#), [estimate\\_bias\(\)](#), [mfrmr\\_visual\\_diagnostics](#), [mfrmr\\_reports\\_and\\_tables](#), [mfrmr\\_reporting\\_and\\_apa](#), [mfrmr\\_linking\\_and\\_dff](#), [mfrmr\\_compatibility\\_layer](#), [summary.mfrm\\_fit\(\)](#), [summary\(diag\)](#), [summary\(\)](#), [plot.mfrm\\_fit\(\)](#), [plot\(\)](#)

**Examples**

```
toy_full <- load_mfrm_data("example_core")
keep_people <- unique(toy_full$Person)[1:12]
toy <- toy_full[toy_full$Person %in% keep_people, , drop = FALSE]

fit <- suppressWarnings(fit_mfrm(
  toy,
  person = "Person",
  facets = c("Rater", "Criterion"),
  score = "Score",
  method = "JML",
  maxit = 10
))
class(summary(fit))

diag <- diagnose_mfrm(fit, residual_pca = "none")
class(summary(diag))

t4 <- unexpected_response_table(fit, diagnostics = diag, top_n = 5)
class(summary(t4))
p <- plot(t4, draw = FALSE)

sc <- subset_connectivity_report(fit, diagnostics = diag)
p_design <- plot(sc, type = "design_matrix", draw = FALSE, preset = "publication")
class(p_design)

chk <- reporting_checklist(fit, diagnostics = diag)
head(chk$checklist[, c("Section", "Item", "DraftReady", "NextAction")])

sim_spec <- build_mfrm_sim_spec(
  n_person = 30,
  n_rater = 4,
  n_criterion = 4,
  raters_per_person = 2,
  assignment = "rotating"
)
pred_pop <- predict_mfrm_population(sim_spec = sim_spec, reps = 2, maxit = 10, seed = 1)
summary(pred_pop)$forecast[, c("Facet", "MeanSeparation", "McseSeparation")]
```

**Description**

List literature-based warning threshold profiles

**Usage**

```
mfrm_threshold_profiles()
```

**Details**

Use this function to inspect available profile presets before calling [build\\_visual\\_summaries\(\)](#).

`profiles` contains thresholds used by warning logic (sample size, fit ratios, PCA cutoffs, etc.).  
`pca_reference_bands` contains literature-oriented descriptive bands used in summary text.

**Value**

An object of class `mfrm_threshold_profiles` with `profiles` (strict, standard, lenient) and `pca_reference_bands`.

**Interpreting output**

- `profiles`: numeric threshold presets (strict, standard, lenient).
- `pca_reference_bands`: narrative reference bands for PCA interpretation.

**Typical workflow**

1. Review presets with `mfrm_threshold_profiles()`.
2. Pick a default profile for project policy.
3. Override only selected fields in [build\\_visual\\_summaries\(\)](#) when needed.

**See Also**

[build\\_visual\\_summaries\(\)](#)

**Examples**

```
profiles <- mfrm_threshold_profiles()
names(profiles)
names(profiles$profiles)
class(profiles)
s_profiles <- summary(profiles)
s_profiles$overview
```

---

plot.apa_table	<i>Plot an APA/FACETS table object using base R</i>
----------------	---

---

### Description

Plot an APA/FACETS table object using base R

### Usage

```
## S3 method for class 'apa_table'  
plot(  
  x,  
  y = NULL,  
  type = c("numeric_profile", "first_numeric"),  
  main = NULL,  
  palette = NULL,  
  label_angle = 45,  
  draw = TRUE,  
  ...  
)
```

### Arguments

x	Output from <a href="#">apa_table()</a> .
y	Reserved for generic compatibility.
type	Plot type: "numeric_profile" (column means) or "first_numeric" (distribution of the first numeric column).
main	Optional title override.
palette	Optional named color overrides.
label_angle	Axis-label rotation angle for bar-type plots.
draw	If TRUE, draw using base graphics.
...	Reserved for generic compatibility.

### Details

Quick visualization helper for numeric columns in [apa\\_table\(\)](#) output. It is intended for table QA and exploratory checks, not final publication graphics.

### Value

A plotting-data object of class `mfrm_plot_data`.

### Interpreting output

- "numeric\_profile": compares column means to spot scale/centering mismatches.
- "first\_numeric": checks distribution shape of the first numeric column.

**Typical workflow**

1. Build table with `apa_table()`.
2. Run `summary(tbl)` for metadata.
3. Use `plot(tbl, type = "numeric_profile")` for quick numeric QC.

**See Also**

`apa_table()`, `summary()`

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
tbl <- apa_table(fit, which = "summary")
p <- plot(tbl, draw = FALSE)
p2 <- plot(tbl, type = "first_numeric", draw = FALSE)
if (interactive()) {
  plot(
    tbl,
    type = "numeric_profile",
    main = "APA Numeric Profile (Customized)",
    palette = c(numeric_profile = "#2b8cbe", grid = "#d9d9d9"),
    label_angle = 45
  )
}
```

---

plot.mfrm\_anchor\_audit

*Plot an anchor-audit object*

---

**Description**

Plot an anchor-audit object

**Usage**

```
## S3 method for class 'mfrm_anchor_audit'
plot(
  x,
  y = NULL,
  type = c("issue_counts", "facet_constraints", "level_observations"),
  main = NULL,
  palette = NULL,
  label_angle = 45,
  draw = TRUE,
  ...
)
```

## Arguments

x	Output from <code>audit_mfrm_anchors()</code> .
y	Reserved for generic compatibility.
type	Plot type: "issue_counts", "facet_constraints", or "level_observations".
main	Optional title override.
palette	Optional named colors.
label_angle	X-axis label angle for bar plots.
draw	If TRUE, draw using base graphics.
...	Reserved for generic compatibility.

## Details

Base-R visualization helper for anchor audit outputs.

## Value

A plotting-data object of class `mfrm_plot_data`.

## Interpreting output

- "issue\_counts": volume of each issue class.
- "facet\_constraints": anchored/grouped/free mix by facet.
- "level\_observations": observation support across levels.

## Typical workflow

1. Run `audit_mfrm_anchors()`.
2. Start with `plot(aud, type = "issue_counts")`.
3. Inspect constraint and support plots before fitting.

## See Also

`audit_mfrm_anchors()`, `make_anchor_table()`

## Examples

```
toy <- load_mfrmr_data("example_core")
aud <- audit_mfrm_anchors(toy, "Person", c("Rater", "Criterion"), "Score")
p <- plot(aud, draw = FALSE)
```

---

plot.mfrm\_bundle      *Plot report/table bundles with base R defaults*

---

### Description

Plot report/table bundles with base R defaults

### Usage

```
## S3 method for class 'mfrm_bundle'
plot(x, y = NULL, type = NULL, ...)
```

### Arguments

x	A bundle object returned by mfrm table/report helpers.
y	Reserved for generic compatibility.
type	Optional plot type. Available values depend on bundle class.
...	Additional arguments forwarded to class-specific plotters.

### Details

plot() dispatches by bundle class:

- mfrm\_unexpected -> [plot\\_unexpected\(\)](#)
- mfrm\_fair\_average -> [plot\\_fair\\_average\(\)](#)
- mfrm\_displacement -> [plot\\_displacement\(\)](#)
- mfrm\_interrater -> [plot\\_interrater\\_agreement\(\)](#)
- mfrm\_facets\_chisq -> [plot\\_facets\\_chisq\(\)](#)
- mfrm\_bias\_interaction -> [plot\\_bias\\_interaction\(\)](#)
- mfrm\_bias\_count -> bias-count plots (cell counts / low-count rates)
- mfrm\_fixed\_reports -> pairwise-contrast diagnostics
- mfrm\_visual\_summaries -> warning/summary message count plots
- mfrm\_category\_structure -> default base-R category plots
- mfrm\_category\_curves -> default ogive/CCC plots
- mfrm\_rating\_scale -> category-counts/threshold plots
- mfrm\_measurable -> measurable-data coverage/count plots
- mfrm\_unexpected\_after\_bias -> post-bias unexpected-response plots
- mfrm\_output\_bundle -> graph/score output-file diagnostics
- mfrm\_residual\_pca -> residual PCA scree/loadings via [plot\\_residual\\_pca\(\)](#)
- mfrm\_specifications -> facet/anchor/convergence plots
- mfrm\_data\_quality -> row-audit/category/missing-row plots

- mfrm\_iteration\_report -> replayed-iteration trajectories
- mfrm\_subset\_connectivity -> subset-observation/connectivity plots
- mfrm\_facet\_statistics -> facet statistic profile plots

If a class is outside these families, use dedicated plotting helpers or custom base R graphics on component tables.

### Value

A plotting-data object of class `mfrm_plot_data`.

### Interpreting output

The returned object is plotting data (`mfrm_plot_data`) that captures the selected route and payload; set `draw = TRUE` for immediate base graphics.

### Typical workflow

1. Create bundle output (e.g., `unexpected_response_table()`).
2. Inspect routing with `summary(bundle)` if needed.
3. Call `plot(bundle, type = ... , draw = FALSE)` to obtain reusable plot data.

### See Also

`summary()`, [plot\\_unexpected\(\)](#), [plot\\_fair\\_average\(\)](#), [plot\\_displacement\(\)](#)

### Examples

```
toy_full <- load_mfrmr_data("example_core")
toy_people <- unique(toy_full$Person)[1:12]
toy <- toy_full[toy_full$Person %in% toy_people, , drop = FALSE]
fit <- suppressWarnings(
  fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 10)
)
t4 <- unexpected_response_table(fit, abs_z_min = 1.5, prob_max = 0.4, top_n = 5)
p <- plot(t4, draw = FALSE)
vis <- build_visual_summaries(fit, diagnose_mfrm(fit, residual_pca = "none"))
p_vis <- plot(vis, type = "comparison", draw = FALSE)
spec <- specifications_report(fit)
p_spec <- plot(spec, type = "facet_elements", draw = FALSE)
if (interactive()) {
  plot(
    t4,
    type = "severity",
    draw = TRUE,
    main = "Unexpected Response Severity (Customized)",
    palette = c(higher = "#d95f02", lower = "#1b9e77", bar = "#2b8cbe"),
    label_angle = 45
  )
  plot(
    vis,
```

```

    type = "comparison",
    draw = TRUE,
    main = "Warning vs Summary Counts (Customized)",
    palette = c(warning = "#cb181d", summary = "#3182bd"),
    label_angle = 45
  )
}

```

---

```
plot.mfrm_data_description
```

*Plot a data-description object*

---

## Description

Plot a data-description object

## Usage

```

## S3 method for class 'mfrm_data_description'
plot(
  x,
  y = NULL,
  type = c("score_distribution", "facet_levels", "missing"),
  main = NULL,
  palette = NULL,
  label_angle = 45,
  draw = TRUE,
  ...
)

```

## Arguments

x	Output from <code>describe_mfrm_data()</code> .
y	Reserved for generic compatibility.
type	Plot type: "score_distribution", "facet_levels", or "missing".
main	Optional title override.
palette	Optional named colors (score, facet, missing).
label_angle	X-axis label angle for bar plots.
draw	If TRUE, draw using base graphics.
...	Reserved for generic compatibility.

## Details

This method draws quick pre-fit quality views from `describe_mfrm_data()`:

- score distribution balance
- facet-level structure size
- missingness by selected columns

## Value

A plotting-data object of class `mfrm_plot_data`.

## Interpreting output

- "score\_distribution": bar chart of weighted observation counts per score category. Y-axis is WeightedN (sum of weights for each category). Categories with very few observations (< 10) may produce unstable threshold estimates. A roughly uniform or unimodal distribution is ideal; heavy floor/ceiling effects compress the measurement range.
- "facet\_levels": bar chart showing the number of distinct levels per facet. Useful for verifying that the design structure matches expectations (e.g., expected number of raters or criteria). Very large numbers of levels increase computation time and may require higher `maxit` in `fit_mfrm()`.
- "missing": bar chart of missing-value counts per input column. Columns with non-zero counts should be investigated before fitting—rows with missing scores, persons, or facet IDs are dropped during estimation.

## Typical workflow

1. Run `describe_mfrm_data()` before fitting.
2. Inspect `summary(ds)` and `plot(ds, type = "missing")`.
3. Check category/facet balance with other plot types.
4. Fit model after resolving obvious data issues.

## See Also

`describe_mfrm_data()`, `plot()`

## Examples

```
toy <- load_mfrm_data("example_core")
ds <- describe_mfrm_data(toy, "Person", c("Rater", "Criterion"), "Score")
p <- plot(ds, draw = FALSE)
```

---

```
plot.mfrm_design_evaluation
      Plot a design-simulation study
```

---

## Description

Plot a design-simulation study

## Usage

```
## S3 method for class 'mfrm_design_evaluation'
plot(
  x,
  facet = c("Rater", "Criterion", "Person"),
  metric = c("separation", "reliability", "infit", "outfit", "misfitrate",
            "severityrmse", "severitybias", "convergencerate", "elapsedsec", "mincategorycount"),
  x_var = c("n_person", "n_rater", "n_criterion", "raters_per_person"),
  group_var = NULL,
  draw = TRUE,
  ...
)
```

## Arguments

x	Output from <code>evaluate_mfrm_design()</code> .
facet	Facet to visualize.
metric	Metric to plot.
x_var	Design variable used on the x-axis.
group_var	Optional design variable used for separate lines.
draw	If TRUE, draw with base graphics; otherwise return plotting data.
...	Reserved for generic compatibility.

## Details

This method is designed for quick design-planning scans rather than polished publication graphics.

Useful first plots are:

- rater metric = "separation" against x\_var = "n\_person"
- criterion metric = "severityrmse" against x\_var = "n\_person" when you want aligned recovery error rather than raw location shifts
- rater metric = "convergencerate" against x\_var = "raters\_per\_person"

## Value

If draw = TRUE, invisibly returns a plotting-data list. If draw = FALSE, returns that list directly.

**See Also**

[evaluate\\_mfrm\\_design\(\)](#), [summary.mfrm\\_design\\_evaluation](#)

**Examples**

```
sim_eval <- evaluate_mfrm_design(  
  n_person = c(30, 50),  
  n_rater = 4,  
  n_criterion = 4,  
  raters_per_person = 2,  
  reps = 1,  
  maxit = 15,  
  seed = 123  
)  
plot(sim_eval, facet = "Rater", metric = "separation", x_var = "n_person", draw = FALSE)
```

---

plot.mfrm\_facets\_run *Plot outputs from a legacy-compatible workflow run*

---

**Description**

Plot outputs from a legacy-compatible workflow run

**Usage**

```
## S3 method for class 'mfrm_facets_run'  
plot(x, y = NULL, type = c("fit", "qc"), ...)
```

**Arguments**

x	A mfrm_facets_run object from <a href="#">run_mfrm_facets()</a> .
y	Unused.
type	Plot route: "fit" delegates to <a href="#">plot.mfrm_fit()</a> and "qc" delegates to <a href="#">plot_qc_dashboard()</a> .
...	Additional arguments passed to the selected plot function.

**Details**

This method is a router for fast visualization from a one-shot workflow result:

- type = "fit" for model-level displays.
- type = "qc" for multi-panel quality-control diagnostics.

**Value**

A plotting object from the delegated plot route.

**Interpreting output**

Returns the plotting object produced by the delegated route: `plot.mfrm_fit()` for "fit" and `plot_qc_dashboard()` for "qc".

**Typical workflow**

1. Run `run_mfrm_facets()`.
2. Start with `plot(out, type = "fit", draw = FALSE)`.
3. Continue with `plot(out, type = "qc", draw = FALSE)` for diagnostics.

**See Also**

[run\\_mfrm\\_facets\(\)](#), [plot.mfrm\\_fit\(\)](#), [plot\\_qc\\_dashboard\(\)](#), [mfrm\\_visual\\_diagnostics](#), [mfrm\\_workflow\\_methods](#)

**Examples**

```
toy <- load_mfrm_data("example_core")
toy_small <- toy[toy$Person %in% unique(toy$Person)[1:12], , drop = FALSE]
out <- run_mfrm_facets(
  data = toy_small,
  person = "Person",
  facets = c("Rater", "Criterion"),
  score = "Score",
  maxit = 10
)
p_fit <- plot(out, type = "fit", draw = FALSE)
class(p_fit)
p_qc <- plot(out, type = "qc", draw = FALSE)
class(p_qc)
```

---

plot.mfrm\_fit

*Plot fitted MFRM results with base R*

---

**Description**

Plot fitted MFRM results with base R

**Usage**

```
## S3 method for class 'mfrm_fit'
plot(
  x,
  type = NULL,
  facet = NULL,
  top_n = 30,
  theta_range = c(-6, 6),
```

```

    theta_points = 241,
    title = NULL,
    palette = NULL,
    label_angle = 45,
    show_ci = FALSE,
    ci_level = 0.95,
    draw = TRUE,
    preset = c("standard", "publication", "compact"),
    ...
  )

```

### Arguments

x	An mfrm_fit object from <code>fit_mfrm()</code> .
type	Plot type. Use NULL, "bundle", or "all" for the three-part fit bundle; otherwise choose one of "facet", "person", "step", "wright", "pathway", or "ccc".
facet	Optional facet name for type = "facet".
top_n	Maximum number of facet/step locations retained for compact displays.
theta_range	Numeric length-2 range for pathway and CCC plots.
theta_points	Number of theta grid points used for pathway and CCC plots.
title	Optional custom title.
palette	Optional color overrides.
label_angle	Rotation angle for x-axis labels where applicable.
show_ci	If TRUE, add approximate confidence intervals when available.
ci_level	Confidence level used when show_ci = TRUE.
draw	If TRUE, draw the plot with base graphics.
preset	Visual preset ("standard", "publication", or "compact").
...	Additional arguments ignored for S3 compatibility.

### Details

This S3 plotting method provides the core fit-family visuals for mfrmr. When type is omitted, it returns a bundle containing a Wright map, pathway map, and category characteristic curves. The returned object still carries machine-readable metadata through the mfrm\_plot\_data contract, even when the plot is drawn immediately.

type = "wright" shows persons, facet levels, and step thresholds on a shared logit scale. type = "pathway" shows expected score traces and dominant-category regions across theta. type = "ccc" shows category response probabilities. The remaining types provide compact person, step, or facet-specific displays.

### Value

Invisibly, an mfrm\_plot\_data object or an mfrm\_plot\_bundle when type is omitted.

**Typical workflow**

1. Fit a model with `fit_mfrm()`.
2. Use `plot(fit)` to inspect the three core fit-family visuals.
3. Switch to `type = "wright"` or `type = "pathway"` when you need a single figure for reporting or manuscript preparation.

**Further guidance**

For a plot-selection guide and extended examples, see [mfrmr\\_visual\\_diagnostics](#) and `vignette("mfrmr-visual-diagnostics", package = "mfrmr")`.

**See Also**

[fit\\_mfrm\(\)](#), [plot\\_wright\\_unified\(\)](#), [plot\\_bubble\(\)](#), [mfrmr\\_visual\\_diagnostics](#)

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(
  toy,
  "Person",
  c("Rater", "Criterion"),
  "Score",
  method = "JML",
  model = "RSM",
  maxit = 25
)
bundle <- plot(fit, draw = FALSE)
names(bundle)
if (interactive()) {
  plot(
    fit,
    type = "wright",
    preset = "publication",
    title = "Customized Wright Map",
    show_ci = TRUE,
    label_angle = 45
  )
  plot(
    fit,
    type = "pathway",
    title = "Customized Pathway Map",
    palette = c("#1f78b4")
  )
  plot(
    fit,
    type = "ccc",
    title = "Customized Category Characteristic Curves",
    palette = c("#1b9e77", "#d95f02", "#7570b3")
  )
}
```

---

`plot.mfrm_signal_detection`*Plot DIF/bias screening simulation results*

---

**Description**

Plot DIF/bias screening simulation results

**Usage**

```
## S3 method for class 'mfrm_signal_detection'
plot(
  x,
  signal = c("dif", "bias"),
  metric = c("power", "false_positive", "estimate", "screen_rate",
            "screen_false_positive"),
  x_var = c("n_person", "n_rater", "n_criterion", "raters_per_person"),
  group_var = NULL,
  draw = TRUE,
  ...
)
```

**Arguments**

<code>x</code>	Output from <a href="#">evaluate_mfrm_signal_detection()</a> .
<code>signal</code>	Whether to plot DIF or bias screening results.
<code>metric</code>	Metric to plot. For <code>signal = "bias"</code> , <code>metric = "power"</code> maps to the screening hit rate ( <code>BiasScreenRate</code> ).
<code>x_var</code>	Design variable used on the x-axis.
<code>group_var</code>	Optional design variable used for separate lines.
<code>draw</code>	If <code>TRUE</code> , draw with base graphics; otherwise return plotting data.
<code>...</code>	Reserved for generic compatibility.

**Value**

If `draw = TRUE`, invisibly returns plotting data. If `draw = FALSE`, returns that plotting-data list directly. The returned list includes `display_metric` and `interpretation_note` so callers can label bias-side plots as screening summaries rather than formal power/error-rate displays.

**See Also**

[evaluate\\_mfrm\\_signal\\_detection\(\)](#), [summary.mfrm\\_signal\\_detection](#)

**Examples**

```
sig_eval <- suppressWarnings(evaluate_mfrm_signal_detection(
  n_person = 20,
  n_rater = 3,
  n_criterion = 3,
  raters_per_person = 2,
  reps = 1,
  maxit = 10,
  bias_max_iter = 1,
  seed = 123
))
plot(sig_eval, signal = "dif", metric = "power", x_var = "n_person", draw = FALSE)
```

---

plot\_anchor\_drift      *Plot anchor drift or a screened linking chain*

---

**Description**

Creates base-R plots for inspecting anchor drift across calibration waves or visualising the cumulative offset in a screened linking chain.

**Usage**

```
plot_anchor_drift(
  x,
  type = c("drift", "chain", "heatmap"),
  facet = NULL,
  preset = c("standard", "publication", "compact"),
  draw = TRUE,
  ...
)
```

**Arguments**

x	An mfrm_anchor_drift or mfrm_equating_chain object.
type	Plot type: "drift" (dot plot of element drift), "chain" (cumulative offset line plot), or "heatmap" (wave-by-element drift heatmap).
facet	Optional character vector to filter drift plots to specific facets.
preset	Visual preset ("standard", "publication", or "compact").
draw	If FALSE, return the plot data invisibly without drawing.
...	Additional graphical parameters passed to base plotting functions.

## Details

Three plot types are supported:

- "drift" (for `mfrm_anchor_drift` objects): A dot plot of each element's drift value, grouped by facet. Horizontal reference lines mark the drift threshold. Red points indicate flagged elements.
- "heatmap" (for `mfrm_anchor_drift` objects): A wave-by-element heat matrix showing drift magnitude. Darker cells represent larger absolute drift. Useful for spotting systematic patterns (e.g., all criteria shifting in the same direction).
- "chain" (for `mfrm_equating_chain` objects): A line plot of cumulative offsets across the screened linking chain. A flatter line indicates smaller between-wave shifts; steep segments suggest larger link offsets that deserve review.

## Value

A plotting-data object of class `mfrm_plot_data`. With `draw = FALSE`, `result$data$table` contains the filtered drift or chain table, `result$data$matrix` contains the heatmap matrix when requested, and the payload includes package-native `title`, `subtitle`, `legend`, and `reference_lines`.

## Which plot should I use?

- Use `type = "drift"` with an `mfrm_anchor_drift` object to review flagged elements directly.
- Use `type = "heatmap"` with an `mfrm_anchor_drift` object to spot wave-by-element patterns.
- Use `type = "chain"` with an `mfrm_equating_chain` object after `build_equating_chain()` to inspect cumulative offsets across waves.

## Interpreting plots

**Drift** is the change in an element's estimated measure between calibration waves, after accounting for the screened common-element link offset. An element is flagged when its absolute drift exceeds a threshold (typically 0.5 logits) **and** the drift-to-SE ratio exceeds a secondary criterion (typically 2.0), ensuring that only practically noticeable and relatively precise shifts are flagged.

- In drift and heatmap plots, red or dark-shaded elements exceed both thresholds. Common causes include rater drift over time, item exposure effects, or curriculum changes.
- In chain plots, uneven spacing between waves suggests differential shifts in the screened linking offsets. The *y*-axis shows cumulative logit-scale offsets; flatter segments indicate more stable adjacent links. Steep segments should be checked alongside `LinkSupportAdequate` and the retained common-element counts before making longitudinal claims.
- For drift objects, it is usually best to read `summary(x)` first and then use the plot to see where the flagged values sit.

## Typical workflow

1. Build a drift or screened-linking object with `detect_anchor_drift()` or `build_equating_chain()`.
2. Start with `draw = FALSE` if you want the plotting data for custom reporting.
3. Use the base-R plot for quick screening and then inspect the underlying tables for exact values.

**Further guidance**

For a plot-selection guide and a longer walkthrough, see [mfrmr\\_visual\\_diagnostics](#) and `vignette("mfrmr-visual-diagnostics", package = "mfrmr")`.

**See Also**

[detect\\_anchor\\_drift\(\)](#), [build\\_equating\\_chain\(\)](#), [plot\\_dif\\_heatmap\(\)](#), [plot\\_bubble\(\)](#), [mfrmr\\_visual\\_diagnostics](#)

**Examples**

```
toy <- load_mfrmr_data("example_core")
people <- unique(toy$Person)
d1 <- toy[toy$Person %in% people[1:12], , drop = FALSE]
d2 <- toy[toy$Person %in% people[13:24], , drop = FALSE]
fit1 <- fit_mfrm(d1, "Person", c("Rater", "Criterion"), "Score",
  method = "JML", maxit = 10)
fit2 <- fit_mfrm(d2, "Person", c("Rater", "Criterion"), "Score",
  method = "JML", maxit = 10)
drift <- detect_anchor_drift(list(W1 = fit1, W2 = fit2))
drift_plot <- plot_anchor_drift(drift, type = "drift", draw = FALSE)
class(drift_plot)
names(drift_plot$data)
chain <- build_equating_chain(list(F1 = fit1, F2 = fit2))
chain_plot <- plot_anchor_drift(chain, type = "chain", draw = FALSE)
head(chain_plot$data$table)
if (interactive()) {
  plot_anchor_drift(drift, type = "heatmap", preset = "publication")
}
```

---

`plot_bias_interaction` *Plot bias interaction diagnostics (preferred alias)*

---

**Description**

Plot bias interaction diagnostics (preferred alias)

**Usage**

```
plot_bias_interaction(
  x,
  plot = c("scatter", "ranked", "abs_t_hist", "facet_profile"),
  diagnostics = NULL,
  facet_a = NULL,
  facet_b = NULL,
  interaction_facets = NULL,
  top_n = 40,
  abs_t_warn = 2,
```

```

  abs_bias_warn = 0.5,
  p_max = 0.05,
  sort_by = c("abs_t", "abs_bias", "prob"),
  main = NULL,
  palette = NULL,
  label_angle = 45,
  preset = c("standard", "publication", "compact"),
  draw = TRUE
)

```

### Arguments

x	Output from <a href="#">estimate_bias()</a> or <a href="#">fit_mfrm()</a> .
plot	Plot type: "scatter", "ranked", "abs_t_hist", or "facet_profile".
diagnostics	Optional output from <a href="#">diagnose_mfrm()</a> (used when x is fit).
facet_a	First facet name (required when x is fit and <code>interaction_facets</code> is not supplied).
facet_b	Second facet name (required when x is fit and <code>interaction_facets</code> is not supplied).
interaction_facets	Character vector of two or more facets.
top_n	Maximum number of ranked rows to keep.
abs_t_warn	Warning cutoff for absolute t statistics.
abs_bias_warn	Warning cutoff for absolute bias size.
p_max	Warning cutoff for p-values.
sort_by	Ranking key: "abs_t", "abs_bias", or "prob".
main	Optional plot title override.
palette	Optional named color overrides (normal, flag, hist, profile).
label_angle	Label angle hint for ranked/profile labels.
preset	Visual preset ("standard", "publication", or "compact").
draw	If TRUE, draw with base graphics.

### Details

Visualization front-end for [bias\\_interaction\\_report\(\)](#) with multiple views.

### Value

A plotting-data object of class `mfrm_plot_data`.

**Plot types**

"scatter" (**default**) Scatter plot of bias size (x) vs screening t-statistic (y). Points colored by flag status. Dashed reference lines at `abs_bias_warn` and `abs_t_warn`. Use for overall triage of interaction effects.

"ranked" Ranked bar chart of top `top_n` interactions sorted by `sort_by` criterion (absolute t, absolute bias, or probability). Bars colored red for flagged cells.

"abs\_t\_hist" Histogram of absolute screening t-statistics across all interaction cells. Dashed reference line at `abs_t_warn`. Use for assessing the overall distribution of interaction effect sizes.

"facet\_profile" Per-facet-level aggregation showing mean absolute bias and flag rate. Useful for identifying which individual facet levels drive systematic interaction patterns.

**Interpreting output**

Start with "scatter" or "ranked" for triage, then confirm pattern shape using "abs\_t\_hist" and "facet\_profile".

Consistent flags across multiple views are stronger screening signals of systematic interaction bias than a single extreme row, but they do not by themselves establish formal inferential evidence.

**Typical workflow**

1. Estimate bias with `estimate_bias()` or pass `mfrm_fit` directly.
2. Plot with `plot = "ranked"` for top interactions.
3. Cross-check using `plot = "scatter"` and `plot = "facet_profile"`.

**See Also**

`bias_interaction_report()`, `estimate_bias()`, `plot_displacement()`

**Examples**

```
toy <- load_mfrmr_data("example_bias")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
p <- plot_bias_interaction(
  fit,
  diagnostics = diagnose_mfrm(fit, residual_pca = "none"),
  facet_a = "Rater",
  facet_b = "Criterion",
  preset = "publication",
  draw = FALSE
)
```

---

plot\_bubble

*Bubble chart of measure estimates and fit statistics*


---

### Description

Produces a Rasch-convention bubble chart where each element is a circle positioned at its measure estimate (x) and fit mean-square (y). Bubble radius reflects approximate measurement precision or sample size.

### Usage

```
plot_bubble(
  x,
  diagnostics = NULL,
  fit_stat = c("Infit", "Outfit"),
  bubble_size = c("SE", "N", "equal"),
  facets = NULL,
  fit_range = c(0.5, 1.5),
  top_n = 60,
  main = NULL,
  palette = NULL,
  draw = TRUE,
  preset = c("standard", "publication", "compact")
)
```

### Arguments

x	Output from <a href="#">fit_mfrm</a> or <a href="#">diagnose_mfrm</a> .
diagnostics	Optional output from <a href="#">diagnose_mfrm</a> when x is an <code>mfrm_fit</code> object. If omitted, diagnostics are computed automatically.
fit_stat	Fit statistic for the y-axis: "Infit" (default) or "Outfit".
bubble_size	Variable controlling bubble radius: "SE" (default), "N" (observation count), or "equal" (uniform size).
facets	Character vector of facets to include. NULL (default) includes all non-person facets.
fit_range	Numeric length-2 vector defining the heuristic fit-review band shown as a shaded region (default <code>c(0.5, 1.5)</code> ).
top_n	Maximum number of elements to plot (default 60).
main	Optional custom plot title.
palette	Optional named colour vector keyed by facet name.
draw	If TRUE (default), render the plot using base graphics.
preset	Visual preset ("standard", "publication", or "compact").

## Details

When `x` is an `mfrm_fit` object and `diagnostics` is omitted, the function computes diagnostics internally via `diagnose_mfrm()`. For repeated plotting in the same workflow, passing a precomputed diagnostics object avoids that extra work.

The x-axis shows element measure estimates on the **logit** scale (one logit = one unit change in log-odds of responding in a higher category). The y-axis shows the selected fit mean-square statistic. A shaded band between `fit_range[1]` and `fit_range[2]` highlights a common heuristic review range.

Bubble radius options:

- "SE": inversely proportional to standard error—larger circles indicate more precisely estimated elements under the current SE approximation.
- "N": proportional to observation count—larger circles indicate elements with more data.
- "equal": uniform size, useful when SE or N differences distract from the fit pattern.

Person estimates are excluded by default because they typically outnumber facet elements and obscure the display.

## Value

Invisibly, an object of class `mfrm_plot_data`.

## Interpreting the plot

Points near the horizontal reference line at 1.0 are closer to model expectation on the selected MnSq scale. Points above 1.5 suggest underfit relative to common review heuristics; these elements may have inconsistent scoring. Points below 0.5 suggest overfit relative to common review heuristics; these may indicate redundancy or restricted range. Points are colored by facet for easy identification.

## Typical workflow

1. Fit a model with `fit_mfrm()`.
2. Compute diagnostics once with `diagnose_mfrm()`.
3. Call `plot_bubble(fit, diagnostics = diag)` to inspect the most extreme elements.

## See Also

[diagnose\\_mfrm](#), [plot\\_unexpected](#), [plot\\_fair\\_average](#)

## Examples

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score",
               method = "JML", model = "RSM", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
plot_bubble(fit, diagnostics = diag, draw = FALSE)
```

---

plot\_dif\_heatmap      *Plot a differential-functioning heatmap*

---

### Description

Visualizes the interaction between a facet and a grouping variable as a heatmap. Rows represent facet levels, columns represent group values, and cell color indicates the selected metric.

### Usage

```
plot_dif_heatmap(x, metric = c("obs_exp", "t", "contrast"), draw = TRUE, ...)
```

### Arguments

x	Output from <a href="#">dif_interaction_table()</a> , <a href="#">analyze_dff()</a> , or <a href="#">analyze_dif()</a> . When an <code>mfrm_dff/mfrm_dif</code> object is passed, the <code>cell_table</code> element is used (requires <code>method = "residual"</code> ).
metric	Which metric to plot: "obs_exp" for observed-minus-expected average (default), "t" for the standardized residual / t-statistic, or "contrast" for pairwise differential-functioning contrast (only for <code>mfrm_dff</code> objects with <code>dif_table</code> ).
draw	If TRUE (default), draw the plot.
...	Additional graphical parameters passed to <a href="#">graphics::image()</a> .

### Value

Invisibly, the matrix used for plotting.

### Interpreting output

- Warm colors (red) indicate positive Obs-Exp values (the model underestimates the facet level for that group).
- Cool colors (blue) indicate negative Obs-Exp values (the model overestimates).
- White/neutral indicates no systematic difference.
- The "contrast" view is best for pairwise differential-functioning summaries, whereas "obs\_exp" and "t" are best for cell-level diagnostics.

### Typical workflow

1. Compute interaction with [dif\\_interaction\\_table\(\)](#) or differential-functioning contrasts with [analyze\\_dff\(\)](#).
2. Plot with `plot_dif_heatmap(...)`.
3. Identify extreme cells or contrasts for follow-up.

### See Also

[dif\\_interaction\\_table\(\)](#), [analyze\\_dff\(\)](#), [analyze\\_dif\(\)](#), [dif\\_report\(\)](#)

**Examples**

```
toy <- load_mfrmr_data("example_bias")

fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score",
               method = "JML", model = "RSM", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
int <- dif_interaction_table(fit, diag, facet = "Rater",
                           group = "Group", data = toy, min_obs = 2)
heat <- plot_dif_heatmap(int, metric = "obs_exp", draw = FALSE)
dim(heat)
```

---

plot\_displacement      *Plot displacement diagnostics using base R*

---

**Description**

Plot displacement diagnostics using base R

**Usage**

```
plot_displacement(
  x,
  diagnostics = NULL,
  anchored_only = FALSE,
  facets = NULL,
  plot_type = c("lollipop", "hist"),
  top_n = 40,
  preset = c("standard", "publication", "compact"),
  draw = TRUE,
  ...
)
```

**Arguments**

x	Output from <code>fit_mfrm()</code> or <code>displacement_table()</code> .
diagnostics	Optional output from <code>diagnose_mfrm()</code> when x is <code>mfrm_fit</code> .
anchored_only	Keep only anchored/group-anchored levels.
facets	Optional subset of facets.
plot_type	"lollipop" or "hist".
top_n	Maximum levels shown in "lollipop" mode.
preset	Visual preset ("standard", "publication", or "compact").
draw	If TRUE, draw with base graphics.
...	Additional arguments passed to <code>displacement_table()</code> when x is <code>mfrm_fit</code> .

## Details

**Displacement** quantifies how much a single element's calibration would shift the overall model if it were allowed to move freely. It is computed as:

$$\text{Displacement}_j = \frac{\sum_i (X_{ij} - E_{ij})}{\sum_i \text{Var}_{ij}}$$

where the sums run over all observations involving element  $j$ . The standard error is  $1/\sqrt{\sum_i \text{Var}_{ij}}$ , and a t-statistic  $t = \text{Displacement}/\text{SE}$  flags elements whose observed residual pattern is inconsistent with the current anchor structure.

Displacement is most informative after anchoring: large values suggest that anchored values may be drifting from the current sample. For non-anchored analyses, displacement reflects residual calibration tension.

## Value

A plotting-data object of class `mfrm_plot_data`.

## Plot types

"lollipop" (**default**) Dot-and-line chart of displacement values. X-axis: displacement (logits). Y-axis: element labels. Points colored red when flagged (default:  $|\text{Disp.}| > 0.5$  logits). Dashed lines at  $\pm$  threshold. Ordered by absolute displacement.

"hist" Histogram of displacement values with Freedman-Diaconis breaks. Dashed reference lines at  $\pm$  threshold. Use for inspecting the overall distribution shape.

## Interpreting output

Lollipop: top absolute displacement levels; flagged points indicate larger movement from anchor expectations.

Histogram: overall displacement distribution and threshold lines. A symmetric distribution centred near zero indicates good anchor stability; heavy tails or skew suggest systematic drift.

Use `anchored_only = TRUE` when your main question is anchor robustness.

## Typical workflow

1. Run with `plot_type = "lollipop"` and `anchored_only = TRUE`.
2. Inspect distribution with `plot_type = "hist"`.
3. Drill into flagged rows via `displacement_table()`.

## Further guidance

For a plot-selection guide and a longer walkthrough, see [mfrmr\\_visual\\_diagnostics](#) and `vignette("mfrmr-visual-diagnos", package = "mfrmr")`.

**See Also**

[displacement\\_table\(\)](#), [plot\\_unexpected\(\)](#), [plot\\_fair\\_average\(\)](#), [plot\\_qc\\_dashboard\(\)](#), [mfrmr\\_visual\\_diagnostics](#)

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
p <- plot_displacement(fit, anchored_only = FALSE, draw = FALSE)
if (interactive()) {
  plot_displacement(
    fit,
    anchored_only = FALSE,
    plot_type = "lollipop",
    preset = "publication"
  )
}
```

---

plot\_facets\_chisq      *Plot facet variability diagnostics using base R*

---

**Description**

Plot facet variability diagnostics using base R

**Usage**

```
plot_facets_chisq(
  x,
  diagnostics = NULL,
  fixed_p_max = 0.05,
  random_p_max = 0.05,
  plot_type = c("fixed", "random", "variance"),
  main = NULL,
  palette = NULL,
  label_angle = 45,
  preset = c("standard", "publication", "compact"),
  draw = TRUE
)
```

**Arguments**

x	Output from <a href="#">fit_mfrm()</a> or <a href="#">facets_chisq_table()</a> .
diagnostics	Optional output from <a href="#">diagnose_mfrm()</a> when x is mfrm_fit.
fixed_p_max	Warning cutoff for fixed-effect chi-square p-values.
random_p_max	Warning cutoff for random-effect chi-square p-values.
plot_type	"fixed", "random", or "variance".

main	Optional custom plot title.
palette	Optional named color overrides (fixed_ok, fixed_flag, random_ok, random_flag, variance).
label_angle	X-axis label angle for bar-style plots.
preset	Visual preset ("standard", "publication", or "compact").
draw	If TRUE, draw with base graphics.

## Details

Facet chi-square tests assess whether the elements within each facet differ significantly.

**Fixed-effect chi-square** tests the null hypothesis  $H_0 : \delta_1 = \delta_2 = \dots = \delta_J$  (all element measures are equal). A flagged result ( $p < \text{fixed\_p\_max}$ ) suggests detectable between-element spread under the fitted model, but it should be interpreted alongside design quality, sample size, and other diagnostics.

**Random-effect chi-square** tests whether element heterogeneity exceeds what would be expected from measurement error alone, treating element measures as random draws. A flagged result is screening evidence that the facet may not be exchangeable under the current model.

**Random variance** is the estimated between-element variance component after removing measurement error. It quantifies the magnitude of true heterogeneity on the logit scale.

## Value

A plotting-data object of class `mfrm_plot_data`.

## Plot types

"fixed" (**default**) Bar chart of fixed-effect chi-square by facet. Bars colored red when the null hypothesis is rejected at `fixed_p_max`. A flagged (red) bar means the facet shows spread worth reviewing under the fitted model.

"random" Bar chart of random-effect chi-square by facet. Bars colored red when rejected at `random_p_max`.

"variance" Bar chart of estimated random variance ( $\text{logit}^2$ ) by facet. Reference line at 0. Larger values indicate greater true heterogeneity among elements.

## Interpreting output

Colored flags reflect configured p-value thresholds (`fixed_p_max`, `random_p_max`). For the fixed test, a flagged (red) result suggests facet spread worth reviewing under the current model. For the random test, a flagged result is screening evidence that the facet may contribute non-trivial heterogeneity beyond measurement error.

## Typical workflow

1. Review "fixed" and "random" panels for flagged facets.
2. Check "variance" to contextualize heterogeneity.
3. Cross-check with inter-rater and element-level fit diagnostics.

**See Also**

[facets\\_chisq\\_table\(\)](#), [plot\\_interrater\\_agreement\(\)](#), [plot\\_qc\\_dashboard\(\)](#)

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
p <- plot_facets_chisq(fit, draw = FALSE)
if (interactive()) {
  plot_facets_chisq(
    fit,
    draw = TRUE,
    plot_type = "fixed",
    preset = "publication",
    main = "Facet Chi-square (Customized)",
    palette = c(fixed_ok = "#2b8cbe", fixed_flag = "#cb181d"),
    label_angle = 45
  )
}
```

---

plot\_facet\_equivalence

*Plot facet-equivalence results*

---

**Description**

Plot facet-equivalence results

**Usage**

```
plot_facet_equivalence(
  x,
  diagnostics = NULL,
  facet = NULL,
  type = c("forest", "rope"),
  draw = TRUE,
  ...
)
```

**Arguments**

x	Output from <a href="#">analyze_facet_equivalence()</a> or <a href="#">fit_mfrm()</a> .
diagnostics	Optional output from <a href="#">diagnose_mfrm()</a> when x is an <code>mfrm_fit</code> object.
facet	Facet to analyze when x is an <code>mfrm_fit</code> object.
type	Plot type: "forest" (default) or "rope".
draw	If TRUE (default), draw the plot. If FALSE, return the prepared plotting data.
...	Additional graphical arguments passed to base plotting functions.

## Details

`plot_facet_equivalence()` is a visual companion to `analyze_facet_equivalence()`. It does not recompute the equivalence analysis; it only reshapes and displays the returned results.

## Value

Invisibly returns the plotting data. If `draw = FALSE`, the plotting data are returned without drawing.

## Plot types

- "forest" places each level on the logit scale with its confidence interval and shades the practical-equivalence region around the weighted grand mean.
- "rope" shows the percentage of each level's uncertainty mass that falls inside the ROPE.

## Interpreting output

In the **forest plot**, the shaded band marks the ROPE ( $\pm$ equivalence\_bound around the weighted grand mean). Levels whose entire confidence interval lies inside this band are close to the facet grand mean under this descriptive screen. Levels whose interval extends outside the band are more displaced from the facet average. Overlapping intervals between two elements suggest they are not reliably separable, but overlap alone does not establish formal equivalence—use the TOST results for that.

In the **ROPE bar chart**, each bar shows the proportion of the element's normal-approximation distribution that falls inside the ROPE-style grand-mean proximity. Values > 95\ the element's normal-approximation uncertainty falls near the facet average; 50–95\ meaningfully displaced from that average.

## Typical workflow

1. Run `analyze_facet_equivalence()`.
2. Start with `type = "forest"` to see the facet on the logit scale.
3. Switch to `type = "rope"` when you want a ranking of levels by grand-mean proximity.

## See Also

[analyze\\_facet\\_equivalence\(\)](#)

## Examples

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score",
               method = "JML", maxit = 25)
eq <- analyze_facet_equivalence(fit, facet = "Rater")
pdat <- plot_facet_equivalence(eq, type = "forest", draw = FALSE)
c(pdat$facet, pdat$type)
```

---

```
plot_facet_quality_dashboard
    Plot a facet-quality dashboard
```

---

### Description

Plot a facet-quality dashboard

### Usage

```
plot_facet_quality_dashboard(
  x,
  diagnostics = NULL,
  facet = NULL,
  bias_results = NULL,
  severity_warn = 1,
  misfit_warn = 1.5,
  central_tendency_max = 0.25,
  bias_count_warn = 1L,
  bias_abs_t_warn = 2,
  bias_abs_size_warn = 0.5,
  bias_p_max = 0.05,
  plot_type = c("severity", "flags"),
  top_n = 20,
  main = NULL,
  palette = NULL,
  label_angle = 45,
  draw = TRUE,
  ...
)
```

### Arguments

x	Output from <code>facet_quality_dashboard()</code> or <code>fit_mfrm()</code> .
diagnostics	Optional output from <code>diagnose_mfrm()</code> when x is a fit.
facet	Optional facet name.
bias_results	Optional bias bundle or list of bundles.
severity_warn	Absolute estimate cutoff used to flag severity outliers.
misfit_warn	Mean-square cutoff used to flag misfit.
central_tendency_max	Absolute estimate cutoff used to flag central tendency.
bias_count_warn	Minimum flagged-bias row count required to flag a level.
bias_abs_t_warn	Absolute t cutoff used when deriving bias-row flags from a raw bias bundle.

bias_abs_size_warn	Absolute bias-size cutoff used when deriving bias-row flags from a raw bias bundle.
bias_p_max	Probability cutoff used when deriving bias-row flags from a raw bias bundle.
plot_type	Plot type, "severity" or "flags".
top_n	Number of rows to keep in the plot data.
main	Optional plot title.
palette	Optional named color overrides.
label_angle	Label angle hint for the "flags" plot.
draw	If TRUE, draw with base graphics.
...	Reserved for generic compatibility.

**Value**

A plotting-data object of class `mfrm_plot_data`.

**See Also**

[facet\\_quality\\_dashboard\(\)](#), [summary.mfrm\\_facet\\_dashboard\(\)](#)

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
p <- plot_facet_quality_dashboard(fit, diagnostics = diag, draw = FALSE)
class(p)
```

---

`plot_fair_average`      *Plot fair-average diagnostics using base R*

---

**Description**

Plot fair-average diagnostics using base R

**Usage**

```
plot_fair_average(
  x,
  diagnostics = NULL,
  facet = NULL,
  metric = c("AdjustedAverage", "StandardizedAdjustedAverage", "FairM", "FairZ"),
  plot_type = c("difference", "scatter"),
  top_n = 40,
  draw = TRUE,
  preset = c("standard", "publication", "compact"),
  ...
)
```

**Arguments**

x	Output from <code>fit_mfrm()</code> or <code>fair_average_table()</code> .
diagnostics	Optional output from <code>diagnose_mfrm()</code> when x is <code>mfrm_fit</code> .
facet	Optional facet name for level-wise lollipop plots.
metric	Adjusted-score metric. Accepts legacy names ("FairM", "FairZ") and package-native names ("AdjustedAverage", "StandardizedAdjustedAverage").
plot_type	"difference" or "scatter".
top_n	Maximum levels shown for "difference" plot.
draw	If TRUE, draw with base graphics.
preset	Visual preset ("standard", "publication", or "compact").
...	Additional arguments passed to <code>fair_average_table()</code> when x is <code>mfrm_fit</code> .

**Details**

Fair-average plots compare observed scoring tendency against model-based fair metrics.

**FairM** is the model-predicted mean score for each element, adjusting for the ability distribution of persons actually encountered. It answers: "What average score would this rater/criterion produce if all raters/criteria saw the same mix of persons?"

**FairZ** standardises FairM to a z-score across elements within each facet, making it easier to compare relative severity across facets with different raw-score scales.

Use FairM when the raw-score metric is meaningful (e.g., reporting average ratings on the original 1–4 scale). Use FairZ when comparing standardised severity ranks across facets.

**Value**

A plotting-data object of class `mfrm_plot_data`. With `draw = FALSE`, the payload includes `title`, `subtitle`, `legend`, `reference_lines`, and the stacked fair-average data.

**Plot types**

"difference" (**default**) Lollipop chart showing the gap between observed and fair-average score for each element. X-axis: Observed - Fair metric. Y-axis: element labels. Points colored teal (lenient, gap  $\geq 0$ ) or orange (severe, gap  $< 0$ ). Ordered by absolute gap.

"scatter" Scatter plot of fair metric (x) vs observed average (y) with an identity line. Points colored by facet. Useful for checking overall alignment between observed and model-adjusted scores.

**Interpreting output**

Difference plot: ranked element-level gaps (Observed - Fair), useful for triage of potentially lenient/severe levels.

Scatter plot: global agreement pattern relative to the identity line.

Larger absolute gaps suggest stronger divergence between observed and model-adjusted scoring.

### Typical workflow

1. Start with `plot_type = "difference"` to find largest discrepancies.
2. Use `plot_type = "scatter"` to check overall alignment pattern.
3. Follow up with facet-level diagnostics for flagged levels.

### Further guidance

For a plot-selection guide and a longer walkthrough, see [mfrmr\\_visual\\_diagnostics](#) and `vignette("mfrmr-visual-diagnostics", package = "mfrmr")`.

### See Also

[fair\\_average\\_table\(\)](#), [plot\\_unexpected\(\)](#), [plot\\_displacement\(\)](#), [plot\\_qc\\_dashboard\(\)](#), [mfrmr\\_visual\\_diagnostics](#)

### Examples

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
p <- plot_fair_average(fit, metric = "AdjustedAverage", draw = FALSE)
if (interactive()) {
  plot_fair_average(fit, metric = "AdjustedAverage", plot_type = "difference")
}
```

---

plot\_information

*Plot design-weighted precision curves*

---

### Description

Visualize the design-weighted precision curve and optionally per-facet-level contribution curves from [compute\\_information\(\)](#).

### Usage

```
plot_information(
  x,
  type = c("tif", "iif", "se", "both"),
  facet = NULL,
  draw = TRUE,
  ...
)
```

**Arguments**

x	Output from <code>compute_information()</code> .
type	"tif" for the overall precision curve (default), "iif" for facet-level contribution curves, "se" for the approximate standard error implied by that curve, or "both" for precision with approximate SE on a secondary axis.
facet	For type = "iif", which facet to plot. If NULL, the first facet is used.
draw	If TRUE (default), draw the plot. If FALSE, return the plot data invisibly.
...	Additional graphical parameters.

**Value**

Invisibly, the plot data (tibble).

**Plot types**

- "tif": overall design-weighted precision across theta.
- "se": approximate standard error across theta.
- "both": precision and approximate SE together, useful for presentations.
- "iif": facet-level contribution curves for one selected facet in an RSM fit.

**Which type should I use?**

- Use "tif" for a quick overall read on precision.
- Use "se" when standard-error language is easier to communicate than precision.
- Use "both" when you want both views in one figure.
- Use "iif" when you want to see which facet levels are shaping the total precision curve.

**Interpreting output**

- The total curve peaks where the realized design is most precise.
- SE is derived as  $1 / \sqrt{\text{precision}}$ ; lower is better.
- Facet-level curves show which facet levels contribute most to that realized precision at each theta.
- If the precision peak sits far from the bulk of person measures, the realized design may be poorly targeted.

**Returned data when draw = FALSE**

For type = "tif", "se", or "both", the returned data come from `x$tif`. For type = "iif", the returned data are the rows of `x$iif` filtered to the requested facet.

**Typical workflow**

1. Compute information with `compute_information()`.
2. Plot with `plot_information(info)` for the total precision curve.
3. Use `plot_information(info, type = "iif", facet = "Rater")` for facet-level contributions.
4. Use `draw = FALSE` when you want the plotting data for custom graphics.

**See Also**

[compute\\_information\(\)](#), [fit\\_mfrm\(\)](#)

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score",
               method = "JML", model = "RSM", maxit = 25)
info <- compute_information(fit)
tif_data <- plot_information(info, type = "tif", draw = FALSE)
head(tif_data)
iif_data <- plot_information(info, type = "iif", facet = "Rater", draw = FALSE)
head(iif_data)
```

---

plot\_interrater\_agreement

*Plot inter-rater agreement diagnostics using base R*

---

**Description**

Plot inter-rater agreement diagnostics using base R

**Usage**

```
plot_interrater_agreement(
  x,
  diagnostics = NULL,
  rater_facet = NULL,
  context_facets = NULL,
  exact_warn = 0.5,
  corr_warn = 0.3,
  plot_type = c("exact", "corr", "difference"),
  top_n = 20,
  main = NULL,
  palette = NULL,
  label_angle = 45,
  preset = c("standard", "publication", "compact"),
  draw = TRUE
)
```

**Arguments**

x	Output from <a href="#">fit_mfrm()</a> or <a href="#">interrater_agreement_table()</a> .
diagnostics	Optional output from <a href="#">diagnose_mfrm()</a> when x is mfrm_fit.
rater_facet	Name of the rater facet when x is mfrm_fit.
context_facets	Optional context facets when x is mfrm_fit.

exact_warn	Warning threshold for exact agreement.
corr_warn	Warning threshold for pairwise correlation.
plot_type	"exact", "corr", or "difference".
top_n	Maximum pairs displayed for bar-style plots.
main	Optional custom plot title.
palette	Optional named color overrides (ok, flag, expected).
label_angle	X-axis label angle for bar-style plots.
preset	Visual preset ("standard", "publication", or "compact").
draw	If TRUE, draw with base graphics.

### Details

Inter-rater agreement plots summarize pairwise consistency for a chosen rater facet. Agreement statistics are computed over observations that share the same person and context-facet levels, ensuring that comparisons reflect identical rating targets.

**Exact agreement** is the proportion of matched observations where both raters assigned the same category score. The **expected agreement** line shows the proportion expected by chance given each rater's marginal category distribution, providing a baseline.

**Pairwise correlation** is the Pearson correlation between scores assigned by each rater pair on matched observations.

The **difference plot** decomposes disagreement into systematic bias (mean signed difference on x-axis: positive = Rater 1 more severe) and total inconsistency (mean absolute difference on y-axis). Points near the origin indicate both low bias and low inconsistency.

The `context_facets` parameter specifies which facets define "the same rating target" (e.g., Criterion). When NULL, all non-rater facets are used as context.

### Value

A plotting-data object of class `mfrm_plot_data`.

### Plot types

"exact" (**default**) Bar chart of exact agreement proportion by rater pair. Expected agreement overlaid as connected circles. Horizontal reference line at `exact_warn`. Bars colored red when observed agreement falls below the warning threshold.

"corr" Bar chart of pairwise Pearson correlation by rater pair. Reference line at `corr_warn`. Ordered by correlation (lowest first). Low correlations suggest inconsistent rank ordering of persons between raters.

"difference" Scatter plot. X-axis: mean signed score difference (Rater 1 – Rater 2); positive values indicate Rater 1 is more severe. Y-axis: mean absolute difference (overall disagreement magnitude). Points colored red when flagged. Vertical reference at 0.

### Interpreting output

Pairs below `exact_warn` and/or `corr_warn` should be prioritized for rater calibration review. On the difference plot, points far from the origin along the x-axis indicate systematic bias; points high on the y-axis indicate large inconsistency regardless of direction.

**Typical workflow**

1. Select rater facet and run "exact" view.
2. Confirm with "corr" view.
3. Use "difference" to inspect directional disagreement.

**Further guidance**

For a plot-selection guide and a longer walkthrough, see [mfrmr\\_visual\\_diagnostics](#) and `vignette("mfrmr-visual-diagnostics", package = "mfrmr")`.

**See Also**

[interrater\\_agreement\\_table\(\)](#), [plot\\_facets\\_chisq\(\)](#), [plot\\_qc\\_dashboard\(\)](#), [mfrmr\\_visual\\_diagnostics](#)

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
p <- plot_interrater_agreement(fit, rater_facet = "Rater", draw = FALSE)
if (interactive()) {
  plot_interrater_agreement(
    fit,
    rater_facet = "Rater",
    draw = TRUE,
    plot_type = "exact",
    main = "Inter-rater Agreement (Customized)",
    palette = c(ok = "#2b8cbe", flag = "#cb181d"),
    label_angle = 45,
    preset = "publication"
  )
}
```

---

plot\_qc\_dashboard

*Plot a base-R QC dashboard*

---

**Description**

Plot a base-R QC dashboard

**Usage**

```
plot_qc_dashboard(
  fit,
  diagnostics = NULL,
  threshold_profile = "standard",
  thresholds = NULL,
  abs_z_min = 2,
```

```

prob_max = 0.3,
rater_facet = NULL,
interrater_exact_warn = 0.5,
interrater_corr_warn = 0.3,
fixed_p_max = 0.05,
random_p_max = 0.05,
top_n = 20,
draw = TRUE,
preset = c("standard", "publication", "compact")
)

```

### Arguments

fit	Output from <code>fit_mfrm()</code> .
diagnostics	Optional output from <code>diagnose_mfrm()</code> .
threshold_profile	Threshold profile name (strict, standard, lenient).
thresholds	Optional named threshold overrides.
abs_z_min	Absolute standardized-residual cutoff for unexpected panel.
prob_max	Maximum observed-category probability cutoff for unexpected panel.
rater_facet	Optional rater facet used in inter-rater panel.
interrater_exact_warn	Warning threshold for inter-rater exact agreement.
interrater_corr_warn	Warning threshold for inter-rater correlation.
fixed_p_max	Warning cutoff for fixed-effect facet chi-square p-values.
random_p_max	Warning cutoff for random-effect facet chi-square p-values.
top_n	Maximum elements displayed in displacement panel.
draw	If TRUE, draw with base graphics.
preset	Visual preset ("standard", "publication", or "compact").

### Details

The dashboard draws nine QC panels in a 3×3 grid:

Panel	What it shows	Key reference lines
1. Category counts	Observed (bars) vs model-expected counts (line)	–
2. Infit vs Outfit	Scatter of element MnSq values	heuristic 0.5, 1.0, 1.5 bands
3.  ZSTD  histogram	Distribution of absolute standardised residuals	ZSTD  = 2
4. Unexpected responses	Standardised residual vs $-\log_{10} P_{\text{obs}}$	abs_z_min, prob_max
5. Fair-average gaps	Boxplots of (Observed - FairM) per facet	zero line
6. Displacement	Top absolute displacement values	±0.5 logits
7. Inter-rater agreement	Exact agreement with expected overlay per pair	interrater_exact_warn
8. Fixed chi-square	Fixed-effect $\chi^2$ per facet	fixed_p_max
9. Separation & Reliability	Bar chart of separation index per facet	–

threshold\_profile controls warning overlays. Three built-in profiles are available: "strict", "standard" (default), and "lenient". Use thresholds to override any profile value with named entries.

### Value

A plotting-data object of class `mfrm_plot_data`.

### Plot types

This function draws a fixed 3×3 panel grid (no `plot_type` argument). For individual panel control, use the dedicated helpers: `plot_unexpected()`, `plot_fair_average()`, `plot_displacement()`, `plot_interrater_agreement()`, `plot_facets_chisq()`.

### Interpreting output

Recommended panel order for fast review:

1. **Category counts + Infit/Outfit** (row 1): first-pass model screening. Category bars should roughly track the expected line; Infit/Outfit points are often reviewed against the heuristic 0.5–1.5 band.
2. **Unexpected responses + Displacement** (row 2): element-level outliers. Sparse points and small displacements are desirable.
3. **Inter-rater + Chi-square** (row 3): facet-level comparability. Read these as screening panels: higher agreement suggests stronger scoring consistency, and significant fixed chi-square indicates detectable facet spread under the current model.
4. **Separation/Reliability** (row 3): approximate screening precision. Higher separation indicates more statistically distinct strata under the current SE approximation.

Treat this dashboard as a screening layer; follow up with dedicated helpers (`plot_unexpected()`, `plot_displacement()`, `plot_interrater_agreement()`, `plot_facets_chisq()`) for detailed diagnosis.

### Typical workflow

1. Fit and diagnose model.
2. Run `plot_qc_dashboard()` for one-page triage.
3. Drill into flagged panels using dedicated functions.

### See Also

`plot_unexpected()`, `plot_fair_average()`, `plot_displacement()`, `plot_interrater_agreement()`, `plot_facets_chisq()`, `build_visual_summaries()`

**Examples**

```

toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
qc <- plot_qc_dashboard(fit, draw = FALSE)
if (interactive()) {
  plot_qc_dashboard(fit, rater_facet = "Rater")
}

```

---

plot\_qc\_pipeline      *Plot QC pipeline results*

---

**Description**

Visualizes the output from `run_qc_pipeline()` as either a traffic-light bar chart or a detail panel showing values versus thresholds.

**Usage**

```
plot_qc_pipeline(x, type = c("traffic_light", "detail"), draw = TRUE, ...)
```

**Arguments**

<code>x</code>	Output from <code>run_qc_pipeline()</code> .
<code>type</code>	Plot type: "traffic_light" (default) or "detail".
<code>draw</code>	If FALSE, return plot data invisibly without drawing.
<code>...</code>	Additional graphical parameters passed to plotting functions.

**Details**

Two plot types are provided for visual triage of QC results:

- "traffic\_light" (default): A horizontal bar chart with one row per QC check. Bars are coloured green (Pass), amber (Warn), or red (Fail). Provides an at-a-glance summary of the current QC review state.
- "detail": A panel showing each check's observed value and its pass/warn/fail thresholds. Useful for understanding how close a borderline result is to the next verdict level.

**Value**

Invisible verdicts tibble from the QC pipeline.

## QC checks performed

The pipeline evaluates up to 10 checks (depending on available diagnostics):

1. **Convergence**: did the optimizer converge?
2. **Overall Infit**: global information-weighted mean-square
3. **Overall Outfit**: global unweighted mean-square
4. **Misfit rate**: proportion of elements with  $|ZSTD| > 2$
5. **Category usage**: minimum observations per score category
6. **Disordered steps**: whether threshold estimates are monotonic
7. **Separation** (per facet): element discrimination adequacy
8. **Residual PCA eigenvalue**: first-component eigenvalue (if computed)
9. **Displacement**: maximum absolute displacement across elements
10. **Inter-rater agreement**: minimum pairwise exact agreement

## Interpreting plots

- **Green** (Pass): the check meets the current threshold-profile criteria.
- **Amber** (Warn): borderline—monitor but not necessarily disqualifying. Review the detail panel to see how close the value is to the fail threshold.
- **Red** (Fail): requires investigation before strong operational or interpretive claims are made from the current run. Common remedies include collapsing categories (for disordered steps), removing outlier raters (for misfit), or increasing sample size (for low separation).
- The detail view shows numeric values, making it easy to communicate exact results to stakeholders.

## See Also

[run\\_qc\\_pipeline\(\)](#), [plot\\_qc\\_dashboard\(\)](#), [build\\_visual\\_summaries\(\)](#), [mfrmr\\_visual\\_diagnostics](#)

## Examples

```
toy <- load_mfrmr_data("study1")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score",
               method = "JML", maxit = 25)
qc <- run_qc_pipeline(fit)
plot_qc_pipeline(qc, draw = FALSE)
```

---

plot\_residual\_pca      *Visualize residual PCA results*

---

### Description

Visualize residual PCA results

### Usage

```
plot_residual_pca(
  x,
  mode = c("overall", "facet"),
  facet = NULL,
  plot_type = c("scree", "loadings"),
  component = 1L,
  top_n = 20L,
  preset = c("standard", "publication", "compact"),
  draw = TRUE
)
```

### Arguments

x	Output from <a href="#">analyze_residual_pca()</a> , <a href="#">diagnose_mfrm()</a> , or <a href="#">fit_mfrm()</a> .
mode	"overall" or "facet".
facet	Facet name for mode = "facet".
plot_type	"scree" or "loadings".
component	Component index for loadings plot.
top_n	Maximum number of variables shown in loadings plot.
preset	Visual preset ("standard", "publication", or "compact").
draw	If TRUE, draws the plot using base graphics.

### Details

x can be either:

- output of [analyze\\_residual\\_pca\(\)](#), or
- a diagnostics object from [diagnose\\_mfrm\(\)](#) (PCA is computed internally), or
- a fitted object from [fit\\_mfrm\(\)](#) (diagnostics and PCA are computed internally).

Plot types:

- "scree": component vs eigenvalue line plot
- "loadings": horizontal bar chart of top absolute loadings

For mode = "facet" and facet = NULL, the first available facet is used.

**Value**

A named list of plotting data (class `mfrm_plot_data`) with:

- `plot`: "scree" or "loadings"
- `mode`: "overall" or "facet"
- `facet`: facet name (or NULL)
- `title`: plot title text
- `data`: underlying table used for plotting

**Interpreting output**

- `plot_type = "scree"`: look for dominant early components relative to later components and the unit-eigenvalue reference line. Treat this as exploratory residual-structure screening, not a standalone unidimensionality test.
- `plot_type = "loadings"`: identifies variables/elements driving each component; inspect both sign and absolute magnitude.

Facet mode (`mode = "facet"`) helps localize residual structure to a specific facet after global PCA review.

**Typical workflow**

1. Run `diagnose_mfrm()` with `residual_pca = "overall"` or "both".
2. Build PCA object via `analyze_residual_pca()` (or pass diagnostics directly).
3. Use scree plot first, then loadings plot for targeted interpretation.

**See Also**

[analyze\\_residual\\_pca\(\)](#), [diagnose\\_mfrm\(\)](#)

**Examples**

```
toy_full <- load_mfrm_data("example_core")
toy_people <- unique(toy_full$Person)[1:24]
toy <- toy_full[match(toy_full$Person, toy_people, nomatch = 0L) > 0L, , drop = FALSE]
fit <- suppressWarnings(
  fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 15)
)
diag <- diagnose_mfrm(fit, residual_pca = "overall")
pca <- analyze_residual_pca(diag, mode = "overall")
plt <- plot_residual_pca(pca, mode = "overall", plot_type = "scree", draw = FALSE)
head(plt$data)
plt_load <- plot_residual_pca(
  pca, mode = "overall", plot_type = "loadings", component = 1, draw = FALSE
)
head(plt_load$data)
if (interactive()) {
  plot_residual_pca(pca, mode = "overall", plot_type = "scree", preset = "publication")
}
```

---

plot\_unexpected      *Plot unexpected responses using base R*

---

## Description

Plot unexpected responses using base R

## Usage

```
plot_unexpected(
  x,
  diagnostics = NULL,
  abs_z_min = 2,
  prob_max = 0.3,
  top_n = 100,
  rule = c("either", "both"),
  plot_type = c("scatter", "severity"),
  main = NULL,
  palette = NULL,
  label_angle = 45,
  preset = c("standard", "publication", "compact"),
  draw = TRUE
)
```

## Arguments

x	Output from <a href="#">fit_mfrm()</a> or <a href="#">unexpected_response_table()</a> .
diagnostics	Optional output from <a href="#">diagnose_mfrm()</a> when x is <code>mfrm_fit</code> .
abs_z_min	Absolute standardized-residual cutoff.
prob_max	Maximum observed-category probability cutoff.
top_n	Maximum rows used from the unexpected table.
rule	Flagging rule ("either" or "both").
plot_type	"scatter" or "severity".
main	Optional custom plot title.
palette	Optional named color overrides (higher, lower, bar).
label_angle	X-axis label angle for "severity" bar plot.
preset	Visual preset ("standard", "publication", or "compact").
draw	If TRUE, draw with base graphics.

## Details

This helper visualizes flagged observations from `unexpected_response_table()`. An observation is "unexpected" when its standardised residual and/or observed-category probability exceed user-specified cutoffs.

The **severity index** is a composite ranking metric that combines the absolute standardised residual  $|Z|$  and the negative log probability  $-\log_{10} P_{\text{obs}}$ . Higher severity indicates responses that are more surprising under the fitted model.

The rule parameter controls flagging logic:

- "either": flag if  $|Z| \geq \text{abs\_z\_min}$  **or**  $P_{\text{obs}} \leq \text{prob\_max}$ .
- "both": flag only if **both** conditions hold simultaneously.

Under common thresholds, many well-behaved runs will produce relatively few flagged observations, but the flagged proportion is design- and model-dependent. Treat the output as a screening display rather than a calibrated goodness-of-fit test.

## Value

A plotting-data object of class `mfrm_plot_data`.

## Plot types

"scatter" (**default**) X-axis: standardized residual  $Z$ . Y-axis:  $-\log_{10}(P_{\text{obs}})$  (negative log of observed-category probability; higher = more surprising). Points colored orange when the observed score is *higher* than expected, teal when *lower*. Dashed lines mark `abs_z_min` and `prob_max` thresholds. Clusters of points in the upper corners indicate systematic misfit patterns worth investigating.

"severity" Ranked bar chart of the composite severity index for the `top_n` most unexpected responses. Bar length reflects the combined unexpectedness; labels identify the specific person-facet combination. Use for QC triage and case-level prioritization.

## Interpreting output

Scatter plot: farther from zero on x-axis = larger residual mismatch; higher y-axis = lower observed-category probability. A uniform scatter with few points beyond the threshold lines indicates fewer locally surprising responses under the current thresholds.

Severity plot: focuses on the most extreme observations for targeted case review. Look for recurring persons or facet levels among the top entries—repeated appearances may signal rater misuse, scoring errors, or model misspecification.

## Typical workflow

1. Fit model and run `diagnose_mfrm()`.
2. Start with "scatter" to assess global unexpected pattern.
3. Switch to "severity" for case prioritization.

**Further guidance**

For a plot-selection guide and a longer walkthrough, see [mfrmr\\_visual\\_diagnostics](#) and `vignette("mfrmr-visual-diagnosis", package = "mfrmr")`.

**See Also**

[unexpected\\_response\\_table\(\)](#), [plot\\_fair\\_average\(\)](#), [plot\\_displacement\(\)](#), [plot\\_qc\\_dashboard\(\)](#), [mfrmr\\_visual\\_diagnostics](#)

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
p <- plot_unexpected(fit, abs_z_min = 1.5, prob_max = 0.4, top_n = 10, draw = FALSE)
if (interactive()) {
  plot_unexpected(
    fit,
    abs_z_min = 1.5,
    prob_max = 0.4,
    top_n = 10,
    plot_type = "severity",
    preset = "publication",
    main = "Unexpected Response Severity (Customized)",
    palette = c(higher = "#d95f02", lower = "#1b9e77", bar = "#2b8cbe"),
    label_angle = 45
  )
}
```

---

plot\_wright\_unified     *Plot a unified Wright map with all facets on a shared logit scale*

---

**Description**

Produces a shared-logit variable map showing person ability distribution alongside measure estimates for every facet in side-by-side columns on the same scale.

**Usage**

```
plot_wright_unified(
  fit,
  diagnostics = NULL,
  bins = 20L,
  show_thresholds = TRUE,
  top_n = 30L,
  show_ci = FALSE,
  ci_level = 0.95,
  draw = TRUE,
  preset = c("standard", "publication", "compact"),
```

```

    palette = NULL,
    label_angle = 45,
    ...
)

```

### Arguments

fit	Output from <code>fit_mfrm()</code> .
diagnostics	Optional output from <code>diagnose_mfrm()</code> .
bins	Integer number of bins for the person histogram. Default 20.
show_thresholds	Logical; if TRUE, display threshold/step positions on the map. Default TRUE.
top_n	Maximum number of facet/step points retained for labeling.
show_ci	Logical; if TRUE, draw approximate confidence intervals when standard errors are available.
ci_level	Confidence level used when <code>show_ci = TRUE</code> .
draw	If TRUE (default), draw the plot. If FALSE, return plot data invisibly.
preset	Visual preset ("standard", "publication", "compact").
palette	Optional named color overrides passed to the shared Wright-map drawer.
label_angle	Rotation angle for group labels on the facet panel.
...	Additional graphical parameters.

### Details

This unified map arranges:

- Column 1: Person measure distribution (horizontal histogram)
- Shared facet/step panel: facet levels and optional threshold positions on the same vertical logit axis
- Range and interquartile overlays for each facet group to show spread

This is the package's most compact targeting view when you want one display that shows where persons, facet levels, and category thresholds sit relative to the same latent scale.

The logit scale on the y-axis is shared, allowing direct visual comparison of all facets and persons.

### Value

Invisibly, a list with persons, facets, and thresholds data used for the plot.

### Interpreting output

- Facet levels at the same height on the map are at similar difficulty.
- The person histogram shows where examinees cluster relative to the facet scale.
- Thresholds (if shown) indicate category boundary positions.
- Large gaps between the person distribution and facet locations can signal targeting problems.

**Typical workflow**

1. Fit a model with `fit_mfrm()`.
2. Plot with `plot_wright_unified(fit)`.
3. Compare person distribution with facet level locations.
4. Use `show_thresholds = TRUE` when you want the category structure in the same view.

**When to use this instead of plot\_information**

Use `plot_wright_unified()` when your main question is targeting or coverage on the shared logit scale. Use `plot_information()` when your main question is measurement precision across theta.

**Further guidance**

For a plot-selection guide and a longer walkthrough, see [mfrmr\\_visual\\_diagnostics](#) and `vignette("mfrmr-visual-diagnosis", package = "mfrmr")`.

**See Also**

[fit\\_mfrm\(\)](#), [plot.mfrm\\_fit\(\)](#), [mfrmr\\_visual\\_diagnostics](#)

**Examples**

```
toy <- load_mfrm_data("example_core")
toy_small <- toy[toy$Person %in% unique(toy$Person)[1:12], , drop = FALSE]
fit <- fit_mfrm(toy_small, "Person", c("Rater", "Criterion"), "Score",
               method = "JML", model = "RSM", maxit = 10)
map_data <- plot_wright_unified(fit, draw = FALSE)
names(map_data)
```

---

```
precision_audit_report
```

*Build a precision audit report*

---

**Description**

Build a precision audit report

**Usage**

```
precision_audit_report(fit, diagnostics = NULL)
```

**Arguments**

<code>fit</code>	Output from <a href="#">fit_mfrm()</a> .
<code>diagnostics</code>	Optional output from <a href="#">diagnose_mfrm()</a> .

## Details

This helper summarizes how `mfrm` derived SE, CI, and reliability values for the current run. It is package-native and is intended to help users distinguish model-based precision paths from exploratory ones without requiring external software conventions.

## Value

A named list with:

- `profile`: one-row precision overview
- `checks`: package-native precision audit checks
- `approximation_notes`: detailed method notes
- `settings`: resolved model and method labels

## What this audit means

`precision_audit_report()` is a reporting gatekeeper for precision claims. It tells you how the package derived uncertainty summaries for the current run and how cautiously those summaries should be written up.

## What this audit does not justify

- It does not, by itself, validate the measurement model or substantive conclusions.
- A favorable precision tier does not override convergence, fit, linking, or design problems elsewhere in the analysis.

## Interpreting output

- `profile`: one-row overview of the active precision tier and recommended use.
- `checks`: package-native audit checks for SE ordering, reliability ordering, coverage of sample/population summaries, and SE source labels.
- `approximation_notes`: method notes copied from `diagnose_mfrm()`.

## Recommended next step

Use the `profile$PrecisionTier` and `checks` table to decide whether SE, CI, and reliability language can be phrased as model-based, should be qualified as hybrid, or should remain exploratory in the final report.

## Typical workflow

1. Run `diagnose_mfrm()` for the fitted model.
2. Build `precision_audit_report(fit, diagnostics = diag)`.
3. Use `summary()` to see whether the run supports model-based reporting language or should remain in exploratory/screening mode.

**See Also**

[diagnose\\_mfrm\(\)](#), [facet\\_statistics\\_report\(\)](#), [reporting\\_checklist\(\)](#)

**Examples**

```
toy <- load_mfrm_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
out <- precision_audit_report(fit, diagnostics = diag)
summary(out)
```

---

predict\_mfrm\_population

*Forecast population-level MFRM operating characteristics for one future design*

---

**Description**

Forecast population-level MFRM operating characteristics for one future design

**Usage**

```
predict_mfrm_population(
  fit = NULL,
  sim_spec = NULL,
  n_person = NULL,
  n_rater = NULL,
  n_criterion = NULL,
  raters_per_person = NULL,
  reps = 50,
  fit_method = NULL,
  model = NULL,
  maxit = 25,
  quad_points = 7,
  residual_pca = c("none", "overall", "facet", "both"),
  seed = NULL
)
```

**Arguments**

fit	Optional output from <a href="#">fit_mfrm()</a> used to derive a fit-based simulation specification.
sim_spec	Optional output from <a href="#">build_mfrm_sim_spec()</a> or <a href="#">extract_mfrm_sim_spec()</a> . Supply exactly one of fit or sim_spec.
n_person	Number of persons/respondents in the future design. Defaults to the value stored in the base simulation specification.

n_rater	Number of rater facet levels in the future design. Defaults to the value stored in the base simulation specification.
n_criterion	Number of criterion/item facet levels in the future design. Defaults to the value stored in the base simulation specification.
raters_per_person	Number of raters assigned to each person in the future design. Defaults to the value stored in the base simulation specification.
reps	Number of replications used in the forecast simulation.
fit_method	Estimation method used inside the forecast simulation. When <code>fit</code> is supplied, defaults to that fit's estimation method; otherwise defaults to "MML".
model	Measurement model used when refitting the forecasted design. Defaults to the model recorded in the base simulation specification.
maxit	Maximum iterations passed to <code>fit_mfrm()</code> in each replication.
quad_points	Quadrature points for <code>fit_method = "MML"</code> .
residual_pca	Residual PCA mode passed to <code>diagnose_mfrm()</code> .
seed	Optional seed for reproducible replications.

## Details

`predict_mfrm_population()` is a **scenario-level forecasting helper** built on top of `evaluate_mfrm_design()`. It is intended for questions such as:

- what separation/reliability would we expect if the next administration had 60 persons, 4 raters, and 2 ratings per person?
- how much Monte Carlo uncertainty remains around those expected summaries?

The function deliberately returns **aggregate operating characteristics** (for example mean separation, reliability, recovery RMSE, convergence rate) rather than future individual true values for one respondent or one rater.

If `fit` is supplied, the function first constructs a fit-derived parametric starting point with `extract_mfrm_sim_spec()` and then evaluates the requested future design under that explicit data-generating mechanism. This should be interpreted as a fit-based forecast under modeling assumptions, not as a guaranteed out-of-sample prediction.

## Value

An object of class `mfrm_population_prediction` with components:

- `design`: requested future design
- `forecast`: facet-level forecast table
- `overview`: run-level overview
- `simulation`: underlying `evaluate_mfrm_design()` result
- `sim_spec`: simulation specification used for the forecast
- `settings`: forecasting settings
- `ademp`: simulation-study metadata
- `notes`: interpretation notes

**Interpreting output**

- forecast contains facet-level expected summaries for the requested future design.
- Mcse\* columns quantify Monte Carlo uncertainty from using a finite number of replications.
- simulation stores the full design-evaluation object in case you want to inspect replicate-level behavior.

**What this does not justify**

This helper does not produce definitive future person measures or rater severities for one concrete sample. It forecasts design-level behavior under the supplied or derived parametric assumptions.

**References**

The forecast is implemented as a one-scenario Monte Carlo / operating- characteristic study following the general guidance of Morris, White, and Crowther (2019) and the ADEMP-oriented reporting framework discussed by Siepe et al. (2024). In *mfrm*, this function is a practical wrapper for future-design planning rather than a direct implementation of a published many-facet forecasting procedure.

- Morris, T. P., White, I. R., & Crowther, M. J. (2019). *Using simulation studies to evaluate statistical methods*. *Statistics in Medicine*, 38(11), 2074-2102.
- Siepe, B. S., Bartos, F., Morris, T. P., Boulesteix, A.-L., Heck, D. W., & Pawel, S. (2024). *Simulation studies for methodological research in psychology: A standardized template for planning, preregistration, and reporting*. *Psychological Methods*.

**See Also**

[build\\_mfrm\\_sim\\_spec\(\)](#), [extract\\_mfrm\\_sim\\_spec\(\)](#), [evaluate\\_mfrm\\_design\(\)](#), [summary.mfrm\\_population\\_prediction](#)

**Examples**

```
spec <- build_mfrm_sim_spec(
  n_person = 40,
  n_rater = 4,
  n_criterion = 4,
  raters_per_person = 2,
  assignment = "rotating"
)
pred <- predict_mfrm_population(
  sim_spec = spec,
  n_person = 60,
  reps = 2,
  maxit = 10,
  seed = 123
)
s_pred <- summary(pred)
s_pred$forecast[, c("Facet", "MeanSeparation", "McseSeparation")]
```

---

predict\_mfrm\_units      *Score future or partially observed units under a fixed MML calibration*

---

### Description

Score future or partially observed units under a fixed MML calibration

### Usage

```
predict_mfrm_units(
  fit,
  new_data,
  person = NULL,
  facets = NULL,
  score = NULL,
  weight = NULL,
  interval_level = 0.95,
  n_draws = 0,
  seed = NULL
)
```

### Arguments

fit	Output from <code>fit_mfrm()</code> with <code>method = "MML"</code> .
new_data	Long-format data for the future or partially observed units to be scored.
person	Optional person column in <code>new_data</code> . Defaults to the person column recorded in <code>fit</code> .
facets	Optional facet-column mapping for <code>new_data</code> . Supply either an unnamed character vector in the calibrated facet order or a named vector whose names are the calibrated facet names and whose values are the column names in <code>new_data</code> .
score	Optional score column in <code>new_data</code> . Defaults to the score column recorded in <code>fit</code> .
weight	Optional weight column in <code>new_data</code> . Defaults to the weight column recorded in <code>fit</code> , if any.
interval_level	Posterior interval level returned in Lower/Upper.
n_draws	Optional number of quadrature-grid posterior draws to return per scored person. Use 0 to skip draws.
seed	Optional seed for reproducible posterior draws.

### Details

`predict_mfrm_units()` is the **individual-unit companion** to `predict_mfrm_population()`. It uses a fixed MML calibration and scores new or partially observed persons via Expected A Posteriori (EAP) summaries on the fitted quadrature grid.

This is appropriate for questions such as:

- what posterior location/uncertainty do these partially observed new respondents have under the existing calibration?
- how uncertain are those scores, given the observed response pattern?

All non-person facet levels in `new_data` must already exist in the fitted calibration. The function does **not** recalibrate the model, update facet estimates, or treat overlapping person IDs as the same latent units from the training data. Person IDs in `new_data` are treated as labels for the rows being scored.

When `n_draws > 0`, the returned draws component contains discrete quadrature-grid posterior draws that can be used as approximate plausible values under the fixed calibration. They should be interpreted as posterior uncertainty summaries, not as deterministic future truth values.

### Value

An object of class `mfrm_unit_prediction` with components:

- `estimates`: posterior summaries by person
- `draws`: optional quadrature-grid posterior draws
- `audit`: row-level preparation audit for `new_data`
- `input_data`: cleaned canonical scoring rows retained from `new_data`
- `settings`: scoring settings
- `notes`: interpretation notes

### Interpreting output

- `estimates` contains posterior EAP summaries for each person in `new_data`.
- Lower and Upper are quadrature-grid posterior interval bounds at the requested `interval_level`.
- SD is posterior uncertainty under the fixed MML calibration.
- `draws`, when requested, contains approximate plausible values on the fitted quadrature grid.

### What this does not justify

This helper does not update the original calibration, estimate new non-person facet levels, or produce deterministic future person true values. It scores new response patterns under a fixed calibration model.

### References

The posterior summaries follow the usual MML/EAP scoring framework used in item response modeling under fixed calibrated parameters (for example Bock & Aitkin, 1981). Optional posterior draws are exposed as quadrature-grid plausible-value-style summaries in the spirit of Mislevy (1991), but here they are offered as practical uncertainty summaries for fixed-calibration many-facet scoring rather than as a direct implementation of a published many-facet plausible-values procedure.

- Bock, R. D., & Aitkin, M. (1981). *Marginal maximum likelihood estimation of item parameters: Application of an EM algorithm*. *Psychometrika*, 46(4), 443-459.
- Mislevy, R. J. (1991). *Randomization-based inference about latent variables from complex samples*. *Psychometrika*, 56(2), 177-196.

**See Also**

[predict\\_mfrm\\_population\(\)](#), [fit\\_mfrm\(\)](#), [summary.mfrm\\_unit\\_prediction](#)

**Examples**

```
toy <- load_mfrm_data("example_core")
keep_people <- unique(toy$Person)[1:18]
toy_fit <- suppressWarnings(
  fit_mfrm(
    toy[toy$Person %in% keep_people, , drop = FALSE],
    "Person", c("Rater", "Criterion"), "Score",
    method = "MML",
    quad_points = 5,
    maxit = 15
  )
)
raters <- unique(toy$Rater)[1:2]
criteria <- unique(toy$Criterion)[1:2]
new_units <- data.frame(
  Person = c("NEW01", "NEW01", "NEW02", "NEW02"),
  Rater = c(raters[1], raters[2], raters[1], raters[2]),
  Criterion = c(criteria[1], criteria[2], criteria[1], criteria[2]),
  Score = c(2, 3, 2, 4)
)
pred_units <- predict_mfrm_units(toy_fit, new_units, n_draws = 0)
summary(pred_units)$estimates[, c("Person", "Estimate", "Lower", "Upper")]
```

---

print.mfrm\_apa\_text    *Print APA narrative text with preserved line breaks*

---

**Description**

Print APA narrative text with preserved line breaks

**Usage**

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

**Arguments**

x                    Character text object from `build_apa_outputs()`\$report\_text.  
 ...                 Reserved for generic compatibility.

**Details**

Prints APA narrative text with preserved paragraph breaks using `cat()`. This is preferred over bare `print()` when you want readable multi-line report output in the console.

**Value**

The input object (invisibly).

**Interpreting output**

The printed text is the same content stored in `build_apa_outputs(...)$report_text`, but with explicit paragraph breaks.

**Typical workflow**

1. Generate `apa <- build_apa_outputs(...)`.
2. Print readable narrative with `apa$report_text`.
3. Use `summary(apa)` to check completeness before manuscript use.

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "both")
apa <- build_apa_outputs(fit, diag)
apa$report_text
```

---

rating\_scale\_table      *Build a rating-scale diagnostics report*

---

**Description**

Build a rating-scale diagnostics report

**Usage**

```
rating_scale_table(
  fit,
  diagnostics = NULL,
  whexact = FALSE,
  drop_unused = FALSE
)
```

**Arguments**

<code>fit</code>	Output from <code>fit_mfrm()</code> .
<code>diagnostics</code>	Optional output from <code>diagnose_mfrm()</code> .
<code>whexact</code>	Use exact ZSTD transformation for category fit.
<code>drop_unused</code>	If TRUE, remove categories with zero count.

## Details

This helper provides category usage/fit statistics and threshold summaries for reviewing score-category functioning. The category usage portion is a global observed-score screen. In PCM fits with a `step_facet`, threshold diagnostics should be interpreted within each `StepFacet` rather than as one pooled whole-scale verdict.

Typical checks:

- sparse category usage (`Count`, `ExpectedCount`)
- category fit (`Infit`, `Outfit`, `ZStd`)
- threshold ordering within each `StepFacet` (`threshold_table$Estimate`, `GapFromPrev`)

## Value

A named list with:

- `category_table`: category-level counts, expected counts, fit, and ZSTD
- `threshold_table`: model step/threshold estimates
- `summary`: one-row summary (usage and threshold monotonicity)

## Interpreting output

Start with `summary`:

- `UsedCategories` close to total `Categories` suggests that most score categories are represented in the observed data.
- very small `MinCategoryCount` indicates potential instability.
- `ThresholdMonotonic = FALSE` indicates disordered thresholds within at least one threshold set. In PCM fits, inspect `threshold_table` by `StepFacet` before drawing scale-wide conclusions.

Then inspect:

- `category_table` for global category-level misfit/sparsity.
- `threshold_table` for adjacent-step gaps and ordering within each `StepFacet`.

## Typical workflow

1. Fit model: `fit_mfrm()`.
2. Build diagnostics: `diagnose_mfrm()`.
3. Run `rating_scale_table()` and review `summary()`.
4. Use `plot()` to visualize category profile quickly.

## Further guidance

For a plot-selection guide and a longer walkthrough, see [mfrmr\\_visual\\_diagnostics](#) and `vignette("mfrmr-visual-diagnos package = "mfrmr")`.

**Output columns**

The `category_table` data.frame contains:

**Category** Score category value.

**Count, Percent** Observed count and percentage of total.

**AvgPersonMeasure** Mean person measure for respondents in this category.

**Infit, Outfit** Category-level fit statistics.

**InfitZSTD, OutfitZSTD** Standardized fit values.

**ExpectedCount, DiffCount** Expected count and observed-expected difference.

**LowCount** Logical; TRUE if count is below minimum threshold.

**InfitFlag, OutfitFlag, ZSTDFlag** Fit-based warning flags.

The `threshold_table` data.frame contains:

**Step** Step label (e.g., "1-2", "2-3").

**Estimate** Estimated threshold/step difficulty (logits).

**StepFacet** Threshold family identifier when the fit uses facet-specific threshold sets.

**GapFromPrev** Difference from the previous threshold within the same StepFacet when thresholds are facet-specific. Gaps below 1.4 logits may indicate category underuse; gaps above 5.0 may indicate wide unused regions (Linacre, 2002).

**ThresholdMonotonic** Logical flag repeated within each threshold set. For PCM fits, read this within StepFacet, not as a pooled item-bank verdict.

**See Also**

[diagnose\\_mfrm\(\)](#), [measurable\\_summary\\_table\(\)](#), [plot.mfrm\\_fit\(\)](#), [mfrmr\\_visual\\_diagnostics](#)

**Examples**

```
toy <- load_mfrm_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
t8 <- rating_scale_table(fit)
summary(t8)
summary(t8)$summary
p_t8 <- plot(t8, draw = FALSE)
class(p_t8)
```

---

recommend\_mfrm\_design *Recommend a design condition from simulation results*

---

## Description

Recommend a design condition from simulation results

## Usage

```
recommend_mfrm_design(
  x,
  facets = c("Rater", "Criterion"),
  min_separation = 2,
  min_reliability = 0.8,
  max_severity_rmse = 0.5,
  max_misfit_rate = 0.1,
  min_convergence_rate = 1,
  prefer = c("n_person", "raters_per_person", "n_rater", "n_criterion")
)
```

## Arguments

x	Output from <code>evaluate_mfrm_design()</code> or <code>summary.mfrm_design_evaluation()</code> .
facets	Facets that must satisfy the planning thresholds.
min_separation	Minimum acceptable mean separation.
min_reliability	Minimum acceptable mean reliability.
max_severity_rmse	Maximum acceptable severity recovery RMSE.
max_misfit_rate	Maximum acceptable mean misfit rate.
min_convergence_rate	Minimum acceptable convergence rate.
prefer	Ranking priority among design variables. Earlier entries are optimized first when multiple designs pass.

## Details

This helper converts a design-study summary into a simple planning table.

A design is marked as recommended when all requested facets satisfy all selected thresholds simultaneously. If multiple designs pass, the helper returns the smallest one according to `prefer` (by default: fewer persons first, then fewer ratings per person, then fewer raters, then fewer criteria).

**Value**

A list of class `mfrm_design_recommendation` with:

- `facet_table`: facet-level threshold checks
- `design_table`: design-level aggregated checks
- `recommended`: the first passing design after ranking
- `thresholds`: thresholds used in the recommendation

**Typical workflow**

1. Run `evaluate_mfrm_design()`.
2. Review `summary.mfrm_design_evaluation()` and `plot.mfrm_design_evaluation()`.
3. Use `recommend_mfrm_design(...)` to identify the smallest acceptable design.

**See Also**

[evaluate\\_mfrm\\_design\(\)](#), [summary.mfrm\\_design\\_evaluation](#), [plot.mfrm\\_design\\_evaluation](#)

**Examples**

```
sim_eval <- evaluate_mfrm_design(  
  n_person = c(30, 50),  
  n_rater = 4,  
  n_criterion = 4,  
  raters_per_person = 2,  
  reps = 1,  
  maxit = 15,  
  seed = 123  
)  
rec <- recommend_mfrm_design(sim_eval)  
rec$recommended
```

---

reference\_case\_audit *Build a package-native reference audit for report completeness*

---

**Description**

Build a package-native reference audit for report completeness

**Usage**

```
reference_case_audit(
  fit,
  diagnostics = NULL,
  bias_results = NULL,
  reference_profile = c("core", "compatibility"),
  include_metrics = TRUE,
  top_n_attention = 15L
)
```

**Arguments**

<code>fit</code>	Output from <code>fit_mfrm()</code> .
<code>diagnostics</code>	Optional output from <code>diagnose_mfrm()</code> . If omitted, diagnostics are computed internally with <code>residual_pca = "none"</code> .
<code>bias_results</code>	Optional output from <code>estimate_bias()</code> . If omitted and at least two facets exist, a 2-way interaction screen is computed internally.
<code>reference_profile</code>	Audit profile. "core" emphasizes package-native report contracts. "compatibility" exposes the manual-aligned compatibility layer used by <code>facets_parity_report(branch = "facets")</code> .
<code>include_metrics</code>	If TRUE, run numerical consistency checks in addition to schema coverage checks.
<code>top_n_attention</code>	Number of lowest-coverage components to keep in <code>attention_items</code> .

**Details**

This function repackages the internal contract audit into package-native terminology so users can review output completeness without needing external manual/table numbering. It reports:

- component-level schema coverage
- numerical consistency checks for derived report tables
- the highest-priority attention items for follow-up

It is an internal completeness audit for package-native outputs, not an external validation study.

Use `reference_profile = "core"` for ordinary mfrm workflows. Use `reference_profile = "compatibility"` only when you explicitly want to inspect the compatibility layer.

**Value**

An object of class `mfrm_reference_audit`.

**Interpreting output**

- overall: one-row internal audit summary with schema coverage and metric pass rate.
- component\_summary: per-component coverage summary.
- attention\_items: quickest list of components needing review.
- metric\_summary / metric\_checks: numerical consistency status.

**See Also**

[facets\\_parity\\_report\(\)](#), [diagnose\\_mfrm\(\)](#), [build\\_fixed\\_reports\(\)](#)

**Examples**

```
toy <- load_mfrm_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
audit <- reference_case_audit(fit, diagnostics = diag)
summary(audit)
```

---

reference\_case\_benchmark

*Benchmark packaged reference cases*

---

**Description**

Benchmark packaged reference cases

**Usage**

```
reference_case_benchmark(
  cases = c("synthetic_truth", "synthetic_bias_contract", "study1_intercal_pair",
    "study2_intercal_pair", "combined_intercal_pair"),
  method = "MML",
  model = "RSM",
  quad_points = 7,
  maxit = 40,
  reltol = 1e-06
)
```

**Arguments**

cases	Reference cases to run. Defaults to all package-native benchmark cases.
method	Estimation method passed to <a href="#">fit_mfrm()</a> . Defaults to "MML".
model	Model family passed to <a href="#">fit_mfrm()</a> . Defaults to "RSM".
quad_points	Quadrature points for method = "MML".
maxit	Maximum optimizer iterations passed to <a href="#">fit_mfrm()</a> .
reltol	Convergence tolerance passed to <a href="#">fit_mfrm()</a> .

## Details

This function audits `mfrm` against the package's curated internal benchmark cases in three ways:

- `synthetic_truth`: checks whether recovered facet measures align with the known generating values from the package's internal synthetic design.
- `synthetic_bias_contract`: checks whether package-native bias tables and pairwise local comparisons satisfy the identities documented in the bias help workflow.
- `*_itercal_pair`: compares a baseline packaged dataset with its iterative recalibration counterpart to review fit stability, facet-measure alignment, and linking coverage together.

The resulting object is intended as an internal benchmark harness for package QA and regression auditing. It does not by itself establish external validity against FACETS, ConQuest, or published calibration studies, and it does not assume any familiarity with external table numbering or printer layouts.

## Value

An object of class `mfrm_reference_benchmark`.

## Interpreting output

- `overview`: one-row internal-benchmark summary.
- `case_summary`: pass/warn/fail triage by reference case.
- `fit_runs`: fitted-run metadata (fit, precision tier, convergence).
- `design_checks`: exact design recovery checks for each dataset.
- `recovery_checks`: known-truth recovery metrics for the internal synthetic case.
- `bias_checks`: source-backed bias/local-measure identity checks.
- `pair_checks`: paired-dataset stability screens for the iterated cases.
- `linking_checks`: common-element audits for paired calibration datasets.
- `source_profile`: source-backed rules that define the internal benchmark contract.

## Examples

```
bench <- reference_case_benchmark(  
  cases = "synthetic_truth",  
  method = "JML",  
  maxit = 30  
)  
summary(bench)
```

---

reporting\_checklist *Build an auto-filled MFRM reporting checklist*

---

### Description

Build an auto-filled MFRM reporting checklist

### Usage

```
reporting_checklist(  
  fit,  
  diagnostics = NULL,  
  bias_results = NULL,  
  include_references = TRUE  
)
```

### Arguments

fit	Output from <code>fit_mfrm()</code> .
diagnostics	Optional output from <code>diagnose_mfrm()</code> . When NULL, diagnostics are computed with <code>residual_pca = "none"</code> .
bias_results	Optional output from <code>estimate_bias()</code> or a named list of such outputs.
include_references	If TRUE, include a compact reference table in the returned bundle.

### Details

This helper ports the app-level reporting checklist into a package-native bundle. It does not try to judge substantive reporting quality; instead, it checks whether the fitted object and related diagnostics contain the evidence typically reported in MFRM write-ups.

Checklist items are grouped into seven sections:

- Method section
- Global fit
- Facet-level statistics
- Element-level statistics
- Rating scale diagnostics
- Bias/interaction analysis
- Visual displays

The output is designed for manuscript preparation, audit trails, and reproducible reporting workflows.

### Value

A named list with checklist tables. Class: `mfrm_reporting_checklist`.

### What this checklist means

`reporting_checklist()` is a content-availability contract. It tells you which reporting elements are already present in the current analysis objects and which still need to be generated or documented. The primary draft-status column is `DraftReady`; `ReadyForAPA` is retained as a backward-compatible alias.

### What this checklist does not justify

- It is not a single run-level pass/fail decision for publication.
- `DraftReady = TRUE / ReadyForAPA = TRUE` does not certify formal inferential adequacy.
- Missing bias rows may simply mean `bias_results` were not supplied.

### Interpreting output

- `checklist`: one row per reporting item with `Available = TRUE/FALSE`. `DraftReady = TRUE` means the item can be drafted into a report with the package's documented caveats. `ReadyForAPA` is a backward-compatible alias of the same flag; neither field certifies formal inferential adequacy.
- `section_summary`: available items by section.
- `references`: core background references when requested.

### Recommended next step

Review the rows with `Available = FALSE` or `DraftReady = FALSE`, then add the missing diagnostics, bias results, or narrative context before calling `build_apa_outputs()` for draft text generation.

### Typical workflow

1. Fit with `fit_mfrm()`.
2. Compute diagnostics with `diagnose_mfrm()`.
3. Run `reporting_checklist()` to see which reporting elements are already available from the current analysis objects.

### See Also

[build\\_apa\\_outputs\(\)](#), [build\\_visual\\_summaries\(\)](#), [specifications\\_report\(\)](#), [data\\_quality\\_report\(\)](#)

### Examples

```
toy <- load_mfrm_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score",
               method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "both")
chk <- reporting_checklist(fit, diagnostics = diag)
summary(chk)
apa <- build_apa_outputs(fit, diag)
head(chk$checklist[, c("Section", "Item", "DraftReady", "NextAction")])
nchar(apa$report_text)
```

---

run_mfrm_facets	<i>Run a legacy-compatible estimation workflow wrapper</i>
-----------------	--

---

### Description

This helper mirrors `mfrmRFacets.R` behavior as a package API and keeps legacy-compatible defaults (`model = "RSM"`, `method = "JML"`), while allowing users to choose compatible estimation options.

### Usage

```
run_mfrm_facets(  
  data,  
  person = NULL,  
  facets = NULL,  
  score = NULL,  
  weight = NULL,  
  keep_original = FALSE,  
  model = c("RSM", "PCM"),  
  method = c("JML", "JMLE", "MML"),  
  step_facet = NULL,  
  anchors = NULL,  
  group_anchors = NULL,  
  noncenter_facet = "Person",  
  dummy_facets = NULL,  
  positive_facets = NULL,  
  quad_points = 15,  
  maxit = 400,  
  reltol = 1e-06,  
  top_n_interactions = 20L  
)
```

```
mfrmRFacets(  
  data,  
  person = NULL,  
  facets = NULL,  
  score = NULL,  
  weight = NULL,  
  keep_original = FALSE,  
  model = c("RSM", "PCM"),  
  method = c("JML", "JMLE", "MML"),  
  step_facet = NULL,  
  anchors = NULL,  
  group_anchors = NULL,  
  noncenter_facet = "Person",  
  dummy_facets = NULL,  
  positive_facets = NULL,
```

```

quad_points = 15,
maxit = 400,
reltol = 1e-06,
top_n_interactions = 20L
)

```

## Arguments

data	A data.frame in long format.
person	Optional person column name. If NULL, guessed from names.
facets	Optional facet column names. If NULL, inferred from remaining columns after person/score/weight mapping.
score	Optional score column name. If NULL, guessed from names.
weight	Optional weight column name.
keep_original	Passed to <code>fit_mfrm()</code> .
model	MFRM model ("RSM" default, or "PCM").
method	Estimation method ("JML" default; "JMLE" and "MML" also supported).
step_facet	Step facet for PCM mode; passed to <code>fit_mfrm()</code> .
anchors	Optional anchor table (data.frame).
group_anchors	Optional group-anchor table (data.frame).
noncenter_facet	Non-centered facet passed to <code>fit_mfrm()</code> .
dummy_facets	Optional dummy facets fixed at zero.
positive_facets	Optional facets with positive orientation.
quad_points	Quadrature points for MML; passed to <code>fit_mfrm()</code> .
maxit	Maximum optimizer iterations.
reltol	Optimization tolerance.
top_n_interactions	Number of rows for interaction diagnostics.

## Details

`run_mfrm_facets()` is intended as a one-shot workflow helper: fit -> diagnostics -> key report tables. Returned objects can be inspected with `summary()` and `plot()`.

## Value

A list with components:

- fit: `fit_mfrm()` result
- diagnostics: `diagnose_mfrm()` result
- iteration: `estimation_iteration_report()` result
- fair\_average: `fair_average_table()` result

- rating\_scale: `rating_scale_table()` result
- run\_info: run metadata table
- mapping: resolved column mapping

### Estimation-method notes

- method = "JML" (default): legacy-compatible joint estimation route.
- method = "JMLE": explicit JMLE label; internally equivalent to JML route.
- method = "MML": marginal maximum likelihood route using `quad_points`.

model = "PCM" is supported; set `step_facet` when facet-specific step structure is needed.

### Visualization

- `plot(out, type = "fit")` delegates to `plot.mfrm_fit()` and returns fit-level visual bundles (e.g., Wright/pathway/CCC).
- `plot(out, type = "qc")` delegates to `plot_qc_dashboard()` and returns a QC dashboard plot object.

### Interpreting output

Start with `summary(out)`:

- check convergence and iteration count in overview.
- confirm resolved columns in mapping.

Then inspect:

- `out$rating_scale` for category/threshold behavior.
- `out$fair_average` for observed-vs-model scoring tendencies.
- `out$diagnostics` for misfit/reliability/interactions.

### Typical workflow

1. Run `run_mfrm_facets()` with explicit column mapping.
2. Check `summary(out)` and `summary(out$diagnostics)`.
3. Visualize with `plot(out, type = "fit")` and `plot(out, type = "qc")`.
4. Export selected tables for reporting (`out$rating_scale`, `out$fair_average`).

### Preferred route for new analyses

For new scripts, prefer the package-native route: `fit_mfrm()` -> `diagnose_mfrm()` -> `reporting_checklist()` -> `build_apa_outputs()`. Use `run_mfrm_facets()` when you specifically need the legacy-compatible one-shot wrapper.

### See Also

`fit_mfrm()`, `diagnose_mfrm()`, `estimation_iteration_report()`, `fair_average_table()`, `rating_scale_table()`, `mfrmr_visual_diagnostics`, `mfrmr_workflow_methods`, `mfrmr_compatibility_layer`

**Examples**

```

toy <- load_mfrmr_data("example_core")
toy_small <- toy[toy$Person %in% unique(toy$Person)[1:12], , drop = FALSE]

# Legacy-compatible default: RSM + JML
out <- run_mfrmr_facets(
  data = toy_small,
  person = "Person",
  facets = c("Rater", "Criterion"),
  score = "Score",
  maxit = 6
)
names(out)
out$fit$summary[, c("Model", "Method")]
s <- summary(out)
s$overview[, c("Model", "Method", "Converged")]
p_fit <- plot(out, type = "fit", draw = FALSE)
class(p_fit)

# Optional: MML route
if (interactive()) {
  out_mml <- run_mfrmr_facets(
    data = toy_small,
    person = "Person",
    facets = c("Rater", "Criterion"),
    score = "Score",
    method = "MML",
    quad_points = 5,
    maxit = 6
  )
  out_mml$fit$summary[, c("Model", "Method")]
}

```

run\_qc\_pipeline

*Run automated quality control pipeline***Description**

Integrates convergence, model fit, reliability, separation, element misfit, unexpected responses, category structure, connectivity, inter-rater agreement, and DIF/bias into a single pass/warn/fail report.

**Usage**

```

run_qc_pipeline(
  fit,
  diagnostics = NULL,
  threshold_profile = "standard",
  thresholds = NULL,

```

```

    rater_facet = NULL,
    include_bias = TRUE,
    bias_results = NULL
  )

```

### Arguments

`fit` Output from `fit_mfrm()`.

`diagnostics` Output from `diagnose_mfrm()`. Computed automatically if NULL.

`threshold_profile` Threshold preset: "strict", "standard" (default), or "lenient".

`thresholds` Named list to override individual thresholds.

`rater_facet` Character name of the rater facet for inter-rater check (auto-detected if NULL).

`include_bias` If TRUE and bias available in diagnostics, check DIF/bias.

`bias_results` Optional pre-computed bias results from `estimate_bias()`.

### Details

The pipeline evaluates 10 quality checks and assigns a verdict (Pass / Warn / Fail) to each. The overall status is the most severe verdict across all checks. Diagnostics are computed automatically via `diagnose_mfrm()` if not supplied.

Reliability and separation are used here as QC signals. In `mfrm`, `Reliability` / `Separation` are model-based facet indices and `RealReliability` / `RealSeparation` provide more conservative lower bounds. For MML, these rely on model-based `ModelSE` values for non-person facets; for JML, they remain exploratory approximations.

Three threshold presets are available via `threshold_profile`:

Aspect	strict	standard	lenient
Global fit warn	1.3	1.5	1.7
Global fit fail	1.5	2.0	2.5
Reliability pass	0.90	0.80	0.70
Separation pass	3.0	2.0	1.5
Misfit warn (pct)	3	5	10
Unexpected fail	3	5	10
Min cat count	15	10	5
Agreement pass	60	50	40
Bias fail (pct)	5	10	15

Individual thresholds can be overridden via the `thresholds` argument (a named list keyed by the internal threshold names shown above).

### Value

Object of class `mfrm_qc_pipeline` with verdicts, overall status, details, and recommendations.

## QC checks

The 10 checks are:

1. **Convergence:** Did the model converge?
2. **Global fit:** Infit/Outfit MnSq within the current review band.
3. **Reliability:** Minimum non-person facet model reliability index.
4. **Separation:** Minimum non-person facet model separation index.
5. **Element misfit:** Percentage of elements with Infit/Outfit outside the current review band.
6. **Unexpected responses:** Percentage of observations with large standardized residuals.
7. **Category structure:** Minimum category count and threshold ordering.
8. **Connectivity:** All observations in a single connected subset.
9. **Inter-rater agreement:** Exact agreement percentage for the rater facet (if applicable).
10. **Functioning/Bias screen:** Percentage of interaction cells that cross the screening threshold (if interaction results are available).

## Interpreting output

- `$overall`: character string "Pass", "Warn", or "Fail".
- `$verdicts`: tibble with columns Check, Verdict, Value, and Threshold for each of the 10 checks.
- `$details`: character vector of human-readable detail strings.
- `$raw_details`: named list of per-check numeric details for programmatic access.
- `$recommendations`: character vector of actionable suggestions for checks that did not pass.
- `$config`: records the threshold profile and effective thresholds.

## Typical workflow

1. Fit a model: `fit <- fit_mfrm(...)`.
2. Optionally compute diagnostics and bias: `diag <- diagnose_mfrm(fit); bias <- estimate_bias(fit, diag, ...)`.
3. Run the pipeline: `qc <- run_qc_pipeline(fit, diag, bias_results = bias)`.
4. Check `qc$overall` for the headline verdict.
5. Review `qc$verdicts` for per-check details.
6. Follow `qc$recommendations` for remediation.
7. Visualize with `plot_qc_pipeline()`.

## See Also

`diagnose_mfrm()`, `estimate_bias()`, `mfrm_threshold_profiles()`, `plot_qc_pipeline()`, `plot_qc_dashboard()`, `build_visual_summaries()`

**Examples**

```
toy <- load_mfrm_data("study1")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score",
               method = "JML", maxit = 25)
qc <- run_qc_pipeline(fit)
qc
summary(qc)
qc$verdicts
```

---

```
sample_mfrm_plausible_values
```

*Sample approximate plausible values under a fixed MML calibration*

---

**Description**

Sample approximate plausible values under a fixed MML calibration

**Usage**

```
sample_mfrm_plausible_values(
  fit,
  new_data,
  person = NULL,
  facets = NULL,
  score = NULL,
  weight = NULL,
  n_draws = 5,
  interval_level = 0.95,
  seed = NULL
)
```

**Arguments**

fit	Output from <code>fit_mfrm()</code> with <code>method = "MML"</code> .
new_data	Long-format data for the future or partially observed units to be scored.
person	Optional person column in <code>new_data</code> . Defaults to the person column recorded in <code>fit</code> .
facets	Optional facet-column mapping for <code>new_data</code> . Supply either an unnamed character vector in the calibrated facet order or a named vector whose names are the calibrated facet names and whose values are the column names in <code>new_data</code> .
score	Optional score column in <code>new_data</code> . Defaults to the score column recorded in <code>fit</code> .
weight	Optional weight column in <code>new_data</code> . Defaults to the weight column recorded in <code>fit</code> , if any.

n_draws	Number of posterior draws per person. Must be a positive integer.
interval_level	Posterior interval level passed to <code>predict_mfrm_units()</code> for the accompanying EAP summary table.
seed	Optional seed for reproducible posterior draws.

### Details

`sample_mfrm_plausible_values()` is a thin public wrapper around `predict_mfrm_units()` that exposes the fixed-calibration posterior draws as a standalone object. It is useful when downstream workflows want repeated latent-value imputations rather than just one posterior EAP summary.

In the current `mfrm` implementation these are **approximate plausible values** drawn from the fitted quadrature-grid posterior under a fixed MML calibration. They should be interpreted as posterior uncertainty summaries for the scored persons, not as deterministic future truth values and not as a full many-facet plausible-values procedure with additional population modeling.

### Value

An object of class `mfrm_plausible_values` with components:

- `values`: one row per person per draw
- `estimates`: companion posterior EAP summaries
- `audit`: row-preparation audit
- `input_data`: cleaned canonical scoring rows retained from `new_data`
- `settings`: scoring settings
- `notes`: interpretation notes

### Interpreting output

- `values` contains one row per person per draw.
- `estimates` contains the companion posterior EAP summaries from `predict_mfrm_units()`.
- `summary()` reports draw counts and empirical draw summaries by person.

### What this does not justify

This helper does not update the calibration, estimate new non-person facet levels, or provide exact future true values. It samples from the fixed-grid posterior implied by the existing MML fit.

### References

The underlying posterior scoring follows the usual MML/EAP framework of Bock and Aitkin (1981). The interpretation of multiple posterior draws as plausible-value-style summaries follows the general logic discussed by Mislevy (1991), while the current implementation remains a practical fixed- calibration approximation rather than a full published many-facet plausible- values method.

- Bock, R. D., & Aitkin, M. (1981). *Marginal maximum likelihood estimation of item parameters: Application of an EM algorithm*. *Psychometrika*, 46(4), 443-459.
- Mislevy, R. J. (1991). *Randomization-based inference about latent variables from complex samples*. *Psychometrika*, 56(2), 177-196.

**See Also**

[predict\\_mfrm\\_units\(\)](#), [summary.mfrm\\_plausible\\_values](#)

**Examples**

```
toy <- load_mfrm_data("example_core")
keep_people <- unique(toy$Person)[1:18]
toy_fit <- suppressWarnings(
  fit_mfrm(
    toy[toy$Person %in% keep_people, , drop = FALSE],
    "Person", c("Rater", "Criterion"), "Score",
    method = "MML",
    quad_points = 5,
    maxit = 15
  )
)
new_units <- data.frame(
  Person = c("NEW01", "NEW01"),
  Rater = unique(toy$Rater)[1],
  Criterion = unique(toy$Criterion)[1:2],
  Score = c(2, 3)
)
pv <- sample_mfrm_plausible_values(toy_fit, new_units, n_draws = 3, seed = 1)
summary(pv)$draw_summary
```

---

simulate\_mfrm\_data      *Simulate long-format many-facet Rasch data for design studies*

---

**Description**

Simulate long-format many-facet Rasch data for design studies

**Usage**

```
simulate_mfrm_data(
  n_person = 50,
  n_rater = 4,
  n_criterion = 4,
  raters_per_person = n_rater,
  score_levels = 4,
  theta_sd = 1,
  rater_sd = 0.35,
  criterion_sd = 0.25,
  noise_sd = 0,
  step_span = 1.4,
  group_levels = NULL,
  dif_effects = NULL,
  interaction_effects = NULL,
```

```

    seed = NULL,
    model = c("RSM", "PCM"),
    step_facet = "Criterion",
    thresholds = NULL,
    assignment = NULL,
    sim_spec = NULL
  )

```

### Arguments

n_person	Number of persons/respondents.
n_rater	Number of rater facet levels.
n_criterion	Number of criterion/item facet levels.
raters_per_person	Number of raters assigned to each person.
score_levels	Number of ordered score categories.
theta_sd	Standard deviation of simulated person measures.
rater_sd	Standard deviation of simulated rater severities.
criterion_sd	Standard deviation of simulated criterion difficulties.
noise_sd	Optional observation-level noise added to the linear predictor.
step_span	Spread of step thresholds on the logit scale.
group_levels	Optional character vector of group labels. When supplied, a balanced Group column is added to the simulated data.
dif_effects	Optional data.frame describing true group-linked DIF effects. Must include Group, at least one design column such as Criterion, and numeric Effect.
interaction_effects	Optional data.frame describing true non-group interaction effects. Must include at least one design column such as Rater or Criterion, plus numeric Effect.
seed	Optional random seed.
model	Measurement model recorded in the simulation setup.
step_facet	Step facet used when model = "PCM" and threshold values vary across levels. Currently "Criterion" and "Rater" are supported.
thresholds	Optional threshold specification. Use either a numeric vector of common thresholds or a data frame with columns StepFacet, Step/StepIndex, and Estimate.
assignment	Assignment design. "crossed" means every person sees every rater; "rotating" uses a balanced rotating subset; "resampled" reuses person-level rater-assignment profiles stored in sim_spec; "skeleton" reuses an observed response skeleton stored in sim_spec, including optional Group/Weight columns when available. When omitted, the function chooses "crossed" if raters_per_person == n_rater, otherwise "rotating".
sim_spec	Optional output from <a href="#">build_mfrm_sim_spec()</a> or <a href="#">extract_mfrm_sim_spec()</a> . When supplied, it defines the generator setup; direct scalar arguments are treated as legacy inputs and should generally be left at their defaults except for seed.

## Details

This function generates synthetic MFRM data from the Rasch model. The data-generating process is:

1. Draw person abilities:  $\theta_n \sim N(0, \text{theta\_sd}^2)$
2. Draw rater severities:  $\delta_j \sim N(0, \text{rater\_sd}^2)$
3. Draw criterion difficulties:  $\beta_i \sim N(0, \text{criterion\_sd}^2)$
4. Generate evenly-spaced step thresholds spanning  $\pm \text{step\_span}/2$
5. For each observation, compute the linear predictor  $\eta = \theta_n - \delta_j - \beta_i + \epsilon$  where  $\epsilon \sim N(0, \text{noise\_sd}^2)$  (optional)
6. Compute category probabilities under the recorded measurement model (RSM or PCM) and sample the response

Latent-value generation is explicit:

- `latent_distribution = "normal"` draws centered normal person/rater/ criterion values using the supplied standard deviations
- `latent_distribution = "empirical"` resamples centered support values recorded in `sim_spec$empirical_support`

When `dif_effects` is supplied, the specified logit shift is added to  $\eta$  for the focal group on the target facet level, creating a known DIF signal. Similarly, `interaction_effects` injects a known bias into specific facet-level combinations.

The generator targets the common two-facet rating design (persons  $\times$  raters  $\times$  criteria). `raters_per_person` controls the incomplete-block structure: when less than `n_rater`, each person is assigned a rotating subset of raters to keep coverage balanced and reproducible.

Threshold handling is intentionally explicit:

- if `thresholds = NULL`, common equally spaced thresholds are generated from `step_span`
- if `thresholds` is a numeric vector, it is used as one common threshold set
- if `thresholds` is a data frame, threshold values may vary by `StepFacet` (currently `Criterion` or `Rater`)

Assignment handling is also explicit:

- `"crossed"` uses the full person  $\times$  rater  $\times$  criterion design
- `"rotating"` assigns a deterministic rotating subset of raters per person
- `"resampled"` reuses empirical person-level rater profiles stored in `sim_spec$assignment_profiles`, optionally carrying over person-level `Group`
- `"skeleton"` reuses an observed person-by-rater-by-criterion response skeleton stored in `sim_spec$design_skeleton`, optionally carrying over `Group` and `Weight`

For more controlled workflows, build a reusable simulation specification first via `build_mfrm_sim_spec()` or derive one from an observed fit with `extract_mfrm_sim_spec()`, then pass it through `sim_spec`.

Returned data include attributes:

- `mfrm_truth`: simulated true parameters (for parameter-recovery checks)
- `mfrm_truth$signals`: injected DIF and interaction signal tables
- `mfrm_simulation_spec`: generation settings (for reproducibility)

**Value**

A long-format data.frame with core columns Study, Person, Rater, Criterion, and Score. If group labels are simulated or reused from an observed response skeleton, a Group column is included. If a weighted response skeleton is reused, a Weight column is also included.

**Interpreting output**

- Higher theta values in `mfrm_truth$person` indicate higher person measures.
- Higher values in `mfrm_truth$facets$Rater` indicate more severe raters.
- Higher values in `mfrm_truth$facets$Criterion` indicate more difficult criteria.
- `mfrm_truth$signals$dif_effects` and `mfrm_truth$signals$interaction_effects` record any injected detection targets.

**Typical workflow**

1. Generate one design with `simulate_mfrm_data()`.
2. Fit with `fit_mfrm()` and diagnose with `diagnose_mfrm()`.
3. For repeated design studies, use `evaluate_mfrm_design()`.

**See Also**

`evaluate_mfrm_design()`, `fit_mfrm()`, `diagnose_mfrm()`

**Examples**

```
sim <- simulate_mfrm_data(  
  n_person = 40,  
  n_rater = 4,  
  n_criterion = 4,  
  raters_per_person = 2,  
  seed = 123  
)  
head(sim)  
names(attr(sim, "mfrm_truth"))
```

---

specifications\_report *Build a specification summary report (preferred alias)*

---

**Description**

Build a specification summary report (preferred alias)

**Usage**

```
specifications_report(  
  fit,  
  title = NULL,  
  data_file = NULL,  
  output_file = NULL,  
  include_fixed = FALSE  
)
```

**Arguments**

<code>fit</code>	Output from <code>fit_mfrm()</code> .
<code>title</code>	Optional analysis title.
<code>data_file</code>	Optional data-file label (for reporting only).
<code>output_file</code>	Optional output-file label (for reporting only).
<code>include_fixed</code>	If TRUE, include a legacy-compatible fixed-width text block.

**Details**

`summary(out)` is supported through `summary()`. `plot(out)` is dispatched through `plot()` for class `mfrm_specifications` (type = "facet\_elements", "anchor\_constraints", "convergence").

**Value**

A named list with specification-report components. Class: `mfrm_specifications`.

**Interpreting output**

- `header / data_spec`: run identity and model settings.
- `facet_labels`: facet sizes and labels.
- `convergence_control`: optimizer configuration and status.

**Typical workflow**

1. Generate `specifications_report(fit)`.
2. Verify model settings and convergence metadata.
3. Use the output as methods and run-documentation support in reports.

**See Also**

[fit\\_mfrm\(\)](#), [data\\_quality\\_report\(\)](#), [estimation\\_iteration\\_report\(\)](#), [mfrm\\_reports\\_and\\_tables](#), [mfrm\\_compatibility\\_layer](#)

## Examples

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
out <- specifications_report(fit, title = "Toy run")
summary(out)
p_spec <- plot(out, draw = FALSE)
class(p_spec)
```

---

subset\_connectivity\_report

*Build a subset connectivity report (preferred alias)*

---

## Description

Build a subset connectivity report (preferred alias)

## Usage

```
subset_connectivity_report(
  fit,
  diagnostics = NULL,
  top_n_subsets = NULL,
  min_observations = 0
)
```

## Arguments

fit	Output from <code>fit_mfrm()</code> .
diagnostics	Optional output from <code>diagnose_mfrm()</code> .
top_n_subsets	Optional maximum number of subset rows to keep.
min_observations	Minimum observations required to keep a subset row.

## Details

`summary(out)` is supported through `summary()`. `plot(out)` is dispatched through `plot()` for class `mfrm_subset_connectivity` (type = "subset\_observations", "facet\_levels", or "linking\_matrix" / "coverage\_matrix" / "design\_matrix").

## Value

A named list with subset-connectivity components. Class: `mfrm_subset_connectivity`.

## Interpreting output

- `summary`: number and size of connected subsets.
- `subset table`: whether data are fragmented into disconnected components.
- `facet-level columns`: where connectivity bottlenecks occur.

**Typical workflow**

1. Run `subset_connectivity_report(fit)`.
2. Confirm near-single-subset structure when possible.
3. Use results to justify linking/anchoring strategy.

**See Also**

[diagnose\\_mfrm\(\)](#), [measurable\\_summary\\_table\(\)](#), [data\\_quality\\_report\(\)](#), [mfrmr\\_linking\\_and\\_dff](#), [mfrmr\\_visual\\_diagnostics](#)

**Examples**

```
toy <- load_mfrm_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
out <- subset_connectivity_report(fit)
summary(out)
p_sub <- plot(out, draw = FALSE)
p_design <- plot(out, type = "design_matrix", draw = FALSE)
class(p_sub)
class(p_design)
out$summary[, c("Subset", "Observations", "ObservationPercent")]
```

---

<code>summary.apa_table</code>	<i>Summarize an APA/FACETS table object</i>
--------------------------------	---

---

**Description**

Summarize an APA/FACETS table object

**Usage**

```
## S3 method for class 'apa_table'
summary(object, digits = 3, top_n = 8, ...)
```

**Arguments**

<code>object</code>	Output from <a href="#">apa_table()</a> .
<code>digits</code>	Number of digits used for numeric summaries.
<code>top_n</code>	Maximum numeric columns shown in <code>numeric_profile</code> .
<code>...</code>	Reserved for generic compatibility.

**Details**

Compact summary helper for QA of table payloads before manuscript export.

**Value**

An object of class `summary.apa_table`.

**Interpreting output**

- overview: table size/composition and missingness.
- numeric\_profile: quick distribution summary of numeric columns.
- caption/note: text metadata readiness.

**Typical workflow**

1. Build table with `apa_table()`.
2. Run `summary(tbl)` and inspect overview.
3. Use `plot.apa_table()` for quick numeric checks if needed.

**See Also**

`apa_table()`, `plot()`

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
tbl <- apa_table(fit, which = "summary")
summary(tbl)
```

---

summary.mfrm\_anchor\_audit

*Summarize an anchor-audit object*

---

**Description**

Summarize an anchor-audit object

**Usage**

```
## S3 method for class 'mfrm_anchor_audit'
summary(object, digits = 3, top_n = 10, ...)
```

**Arguments**

<code>object</code>	Output from <code>audit_mfrm_anchors()</code> .
<code>digits</code>	Number of digits for numeric rounding.
<code>top_n</code>	Maximum rows shown in issue previews.
<code>...</code>	Reserved for generic compatibility.

**Details**

This summary provides a compact pre-estimation audit of anchor and group-anchor specifications.

**Value**

An object of class `summary.mfrm_anchor_audit`.

**Interpreting output**

Recommended order:

- `issue_counts`: primary triage table (non-zero issues first).
- `facet_summary`: anchored/grouped/free-level balance by facet.
- `level_observation_summary` and `category_counts`: sparse-cell diagnostics.
- `recommendations`: concrete remediation suggestions.

If `issue_counts` is non-empty, treat anchor constraints as provisional and resolve issues before final estimation.

**Typical workflow**

1. Run `audit_mfrm_anchors()` with intended anchors/group anchors.
2. Review `summary(aud)` and recommendations.
3. Revise anchor tables, then call `fit_mfrm()`.

**See Also**

`audit_mfrm_anchors()`, `fit_mfrm()`

**Examples**

```
toy <- load_mfrmr_data("example_core")
aud <- audit_mfrm_anchors(toy, "Person", c("Rater", "Criterion"), "Score")
summary(aud)
```

---

`summary.mfrm_apa_outputs`

*Summarize APA report-output bundles*

---

**Description**

Summarize APA report-output bundles

**Usage**

```
## S3 method for class 'mfrm_apa_outputs'
summary(object, top_n = 3, preview_chars = 160, ...)
```

**Arguments**

object	Output from <a href="#">build_apa_outputs()</a> .
top_n	Maximum non-empty lines shown in each component preview.
preview_chars	Maximum characters shown in each preview cell.
...	Reserved for generic compatibility.

**Details**

This summary is a diagnostics layer for APA text products, not a replacement for the full narrative. It reports component completeness, line/character volume, and a compact preview for quick QA before manuscript insertion.

**Value**

An object of class `summary.mfrm_apa_outputs`.

**Interpreting output**

- overview: total coverage across standard text components.
- components: per-component density and mention checks (including residual-PCA mentions).
- sections: package-native section coverage table.
- content\_checks: contract-based alignment checks for APA drafting readiness.
- overview\$DraftContractPass: the primary contract-completeness flag for draft text components.
- overview\$ReadyForAPA: a backward-compatible alias of that contract flag, not a certification of inferential adequacy.
- preview: first non-empty lines for fast visual review.

**Typical workflow**

1. Build outputs via [build\\_apa\\_outputs\(\)](#).
2. Run `summary(apa)` to screen for empty/short components.
3. Use `apa$report_text`, `apa$table_figure_notes`, and `apa$table_figure_captions` as draft components for final-text review.

**See Also**

[build\\_apa\\_outputs\(\)](#), [summary\(\)](#)

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "both")
apa <- build_apa_outputs(fit, diag)
summary(apa)
```

---

summary.mfrm\_bias      *Summarize an mfrm\_bias object in a user-friendly format*

---

### Description

Summarize an mfrm\_bias object in a user-friendly format

### Usage

```
## S3 method for class 'mfrm_bias'
summary(object, digits = 3, top_n = 10, p_cut = 0.05, ...)
```

### Arguments

object	Output from <code>estimate_bias()</code> .
digits	Number of digits for printed numeric values.
top_n	Number of strongest bias rows to keep.
p_cut	Significance cutoff used for counting flagged rows.
...	Reserved for generic compatibility.

### Details

This method returns a compact interaction-bias summary:

- interaction facets/order and analyzed cell counts
- effect-size profile ( $|bias|$  mean/max, significant cell count)
- fixed-effect chi-square block
- iteration-end convergence indicators
- top rows ranked by absolute  $t$

### Value

An object of class `summary.mfrm_bias` with:

- `overview`: interaction facets/order, cell counts, and effect-size profile
- `chi_sq`: fixed-effect chi-square block
- `final_iteration`: end-of-iteration status row
- `top_rows`: highest- $|t|$  interaction rows
- `notes`: short interpretation notes

### Interpreting output

- `overview`: interaction order, analyzed cells, and effect-size profile.
- `chi_sq`: fixed-effect test block.
- `final_iteration`: end-of-loop status from the bias routine.
- `top_rows`: strongest bias contrasts by  $|t|$ .

**Typical workflow**

1. Estimate interactions with `estimate_bias()`.
2. Check `summary(bias)` for screen-positive and unstable cells.
3. Use `bias_interaction_report()` or `plot_bias_interaction()` for details.

**See Also**

`estimate_bias()`, `bias_interaction_report()`

**Examples**

```
toy <- load_mfrmr_data("example_bias")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
bias <- estimate_bias(fit, diag, facet_a = "Rater", facet_b = "Criterion", max_iter = 2)
summary(bias)
```

---

summary.mfrm\_bundle    *Summarize report/table bundles in a user-friendly format*

---

**Description**

Summarize report/table bundles in a user-friendly format

**Usage**

```
## S3 method for class 'mfrm_bundle'
summary(object, digits = 3, top_n = 10, ...)
```

**Arguments**

<code>object</code>	Any report bundle produced by mfrmr table/report helpers.
<code>digits</code>	Number of digits for printed numeric values.
<code>top_n</code>	Number of preview rows shown from the main table component.
<code>...</code>	Reserved for generic compatibility.

**Details**

This method provides a compact summary for bundle-like outputs (for example: unexpected-response, fair-average, chi-square, and category report objects). It extracts:

- object class and available components
- one-row summary table when available
- preview rows from the main data component
- resolved settings/options

Branch-aware summaries are provided for:

- `mfrm_bias_count` (`branch = "original" / "facets"`)
- `mfrm_fixed_reports` (`branch = "original" / "facets"`)
- `mfrm_visual_summaries` (`branch = "original" / "facets"`)

Additional class-aware summaries are provided for:

- `mfrm_unexpected`, `mfrm_fair_average`, `mfrm_displacement`
- `mfrm_interrater`, `mfrm_facets_chisq`, `mfrm_bias_interaction`
- `mfrm_rating_scale`, `mfrm_category_structure`, `mfrm_category_curves`
- `mfrm_measurable`, `mfrm_unexpected_after_bias`, `mfrm_output_bundle`
- `mfrm_residual_pca`, `mfrm_specifications`, `mfrm_data_quality`
- `mfrm_iteration_report`, `mfrm_subset_connectivity`, `mfrm_facet_statistics`
- `mfrm_parity_report`

## Value

An object of class `summary.mfrm_bundle`.

## Interpreting output

- `overview`: class, component count, and selected preview component.
- `summary`: one-row aggregate block when supplied by the bundle.
- `preview`: first `top_n` rows from the main table-like component.
- `settings`: resolved option values if available.

## Typical workflow

1. Generate a bundle table/report helper output.
2. Run `summary(bundle)` for compact QA.
3. Drill into specific components via `$` and visualize with `plot(bundle, ...)`.

## See Also

[unexpected\\_response\\_table\(\)](#), [fair\\_average\\_table\(\)](#), [plot\(\)](#)

## Examples

```
toy_full <- load_mfrmr_data("example_core")
toy_people <- unique(toy_full$Person)[1:12]
toy <- toy_full[toy_full$Person %in% toy_people, , drop = FALSE]
fit <- suppressWarnings(
  fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 10)
)
t4 <- unexpected_response_table(fit, abs_z_min = 1.5, prob_max = 0.4, top_n = 5)
summary(t4)
```

```
diag <- diagnose_mfrm(fit, residual_pca = "none")
bias <- estimate_bias(fit, diag, facet_a = "Rater", facet_b = "Criterion", max_iter = 2)
t11 <- bias_count_table(bias, branch = "facets")
summary(t11)
```

---

summary.mfrm\_data\_description

*Summarize a data-description object*

---

## Description

Summarize a data-description object

## Usage

```
## S3 method for class 'mfrm_data_description'
summary(object, digits = 3, top_n = 10, ...)
```

## Arguments

object	Output from <code>describe_mfrm_data()</code> .
digits	Number of digits for numeric rounding.
top_n	Maximum rows shown in preview blocks.
...	Reserved for generic compatibility.

## Details

This summary is intended as a compact pre-fit quality snapshot for manuscripts and analysis logs.

## Value

An object of class `summary.mfrm_data_description`.

## Interpreting output

Recommended read order:

- overview: sample size, persons/facets/categories.
- missing: missingness hotspots by selected input columns.
- score\_distribution: category usage balance.
- facet\_overview: coverage per facet (minimum/maximum weighted counts).
- agreement: observed-score inter-rater agreement (when available).

Very low `MinWeightedN` in `facet_overview` is a practical warning for unstable downstream facet estimates.

### Typical workflow

1. Run `describe_mfrm_data()` on raw long-format data.
2. Inspect `summary(ds)` before model fitting.
3. Resolve sparse/missing issues, then run `fit_mfrm()`.

### See Also

`describe_mfrm_data()`, `summary.mfrm_fit()`

### Examples

```
toy <- load_mfrm_data("example_core")
ds <- describe_mfrm_data(toy, "Person", c("Rater", "Criterion"), "Score")
summary(ds)
```

---

```
summary.mfrm_design_evaluation
      Summarize a design-simulation study
```

---

### Description

Summarize a design-simulation study

### Usage

```
## S3 method for class 'mfrm_design_evaluation'
summary(object, digits = 3, ...)
```

### Arguments

<code>object</code>	Output from <code>evaluate_mfrm_design()</code> .
<code>digits</code>	Number of digits used in the returned numeric summaries.
<code>...</code>	Reserved for generic compatibility.

### Details

The summary emphasizes condition-level averages that are useful for practical design planning, especially:

- convergence rate
- separation and reliability by facet
- severity recovery RMSE
- mean misfit rate

**Value**

An object of class `summary.mfrm_design_evaluation` with components:

- `overview`: run-level overview
- `design_summary`: aggregated design-by-facet metrics
- `ademp`: simulation-study metadata carried forward from the original object
- `notes`: short interpretation notes

**See Also**

[evaluate\\_mfrm\\_design\(\)](#), [plot.mfrm\\_design\\_evaluation](#)

**Examples**

```
sim_eval <- evaluate_mfrm_design(  
  n_person = c(30, 50),  
  n_rater = 4,  
  n_criterion = 4,  
  raters_per_person = 2,  
  reps = 1,  
  maxit = 15,  
  seed = 123  
)  
summary(sim_eval)
```

---

`summary.mfrm_diagnostics`

*Summarize an `mfrm_diagnostics` object in a user-friendly format*

---

**Description**

Summarize an `mfrm_diagnostics` object in a user-friendly format

**Usage**

```
## S3 method for class 'mfrm_diagnostics'  
summary(object, digits = 3, top_n = 10, ...)
```

**Arguments**

<code>object</code>	Output from <a href="#">diagnose_mfrm()</a> .
<code>digits</code>	Number of digits for printed numeric values.
<code>top_n</code>	Number of highest-absolute-Z fit rows to keep.
<code>...</code>	Reserved for generic compatibility.

## Details

This method returns a compact diagnostics summary designed for quick review:

- design overview (observations, persons, facets, categories, subsets)
- global fit statistics
- approximate reliability/separation by facet
- top facet/person fit rows by absolute ZSTD
- counts of flagged diagnostics (unexpected, displacement, interactions)

## Value

An object of class `summary.mfrm_diagnostics` with:

- `overview`: design-level counts and residual-PCA mode
- `overall_fit`: global fit block
- `reliability`: facet-level separation/reliability summary
- `top_fit`: top  $|ZSTD|$  rows
- `flags`: compact flag counts for major diagnostics
- `notes`: short interpretation notes

## Interpreting output

- `overview`: analysis scale, subset count, and residual-PCA mode.
- `overall_fit`: global fit indices.
- `reliability`: facet separation/reliability block, including model and real bounds when available.
- `top_fit`: highest  $|ZSTD|$  elements for immediate inspection.
- `flags`: compact counts for key warning domains.

## Typical workflow

1. Run diagnostics with `diagnose_mfrm()`.
2. Review `summary(diag)` for major warnings.
3. Follow up with dedicated tables/plots for flagged domains.

## See Also

[diagnose\\_mfrm\(\)](#), [summary.mfrm\\_fit\(\)](#)

## Examples

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
summary(diag)
```

---

`summary.mfrm_facets_run`*Summarize a legacy-compatible workflow run*

---

## Description

Summarize a legacy-compatible workflow run

## Usage

```
## S3 method for class 'mfrm_facets_run'  
summary(object, digits = 3, top_n = 10, ...)
```

## Arguments

<code>object</code>	Output from <code>run_mfrm_facets()</code> .
<code>digits</code>	Number of digits for numeric rounding in summaries.
<code>top_n</code>	Maximum rows shown in nested preview tables.
<code>...</code>	Passed through to nested summary methods.

## Details

This method returns a compact cross-object summary that combines:

- model overview (`object$fit$summary`)
- resolved column mapping
- run settings (`run_info`)
- nested summaries of fit and diagnostics

## Value

An object of class `summary.mfrm_facets_run`.

## Interpreting output

- overview: convergence, information criteria, and scale size.
- mapping: sanity check for auto/explicit column mapping.
- fit / diagnostics: drill-down summaries for reporting decisions.

## Typical workflow

1. Run `run_mfrm_facets()` to execute a one-shot pipeline.
2. Inspect with `summary(out)` for mapping and convergence checks.
3. Review nested objects (`out$fit`, `out$diagnostics`) as needed.

**See Also**

[run\\_mfrm\\_facets\(\)](#), [summary.mfrm\\_fit\(\)](#), [mfrmr\\_workflow\\_methods](#), [summary\(\)](#)

**Examples**

```
toy <- load_mfrmr_data("example_core")
toy_small <- toy[toy$Person %in% unique(toy$Person)[1:12], , drop = FALSE]
out <- run_mfrm_facets(
  data = toy_small,
  person = "Person",
  facets = c("Rater", "Criterion"),
  score = "Score",
  maxit = 10
)
s <- summary(out)
s$overview[, c("Model", "Method", "Converged")]
s$mapping
```

---

summary.mfrm\_facet\_dashboard

*Summarize a facet-quality dashboard*

---

**Description**

Summarize a facet-quality dashboard

**Usage**

```
## S3 method for class 'mfrm_facet_dashboard'
summary(object, digits = 3, top_n = 10, ...)
```

**Arguments**

object	Output from <a href="#">facet_quality_dashboard()</a> .
digits	Number of digits for printed numeric values.
top_n	Number of flagged levels to preview.
...	Reserved for generic compatibility.

**Value**

An object of class `summary.mfrm_facet_dashboard`.

**See Also**

[facet\\_quality\\_dashboard\(\)](#), [plot\\_facet\\_quality\\_dashboard\(\)](#)

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrm(fit, residual_pca = "none")
summary(facet_quality_dashboard(fit, diagnostics = diag))
```

---

summary.mfrm_fit	<i>Summarize an mfrm_fit object in a user-friendly format</i>
------------------	---

---

**Description**

Summarize an mfrm\_fit object in a user-friendly format

**Usage**

```
## S3 method for class 'mfrm_fit'
summary(object, digits = 3, top_n = 5, ...)
```

**Arguments**

object	Output from <code>fit_mfrm()</code> .
digits	Number of digits for printed numeric values.
top_n	Number of extreme facet/person rows shown in summaries.
...	Reserved for generic compatibility.

**Details**

This method provides a compact, human-readable summary oriented to reporting. It returns a structured object and prints:

- model fit overview (N, LogLik, AIC/BIC, convergence)
- facet-level estimate distribution (mean/SD/range)
- person measure distribution
- step/threshold checks
- high/low person measures and extreme facet levels

**Value**

An object of class `summary.mfrm_fit` with:

- overview: global model/fit indicators
- facet\_overview: per-facet estimate distribution summary
- person\_overview: person-measure distribution summary
- step\_overview: threshold/step diagnostics
- top\_person: highest/lowest person measures
- top\_facet: extreme facet-level estimates
- notes: short interpretation notes

**Interpreting output**

- overview: convergence and information criteria.
- facet\_overview: per-facet spread and range of estimates.
- person\_overview: distribution of person measures.
- step\_overview: threshold spread and monotonicity checks.
- top\_person / top\_facet: extreme estimates for quick triage.

**Typical workflow**

1. Fit model with `fit_mfrm()`.
2. Run `summary(fit)` for first-pass diagnostics.
3. Continue with `diagnose_mfrm()` for element-level fit checks.

**See Also**

`fit_mfrm()`, `diagnose_mfrm()`

**Examples**

```
toy <- load_mfrmr_data("example_core")
fit <- fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
summary(fit)
```

---

```
summary.mfrm_plausible_values
```

*Summarize approximate plausible values from fixed-calibration scoring*

---

**Description**

Summarize approximate plausible values from fixed-calibration scoring

**Usage**

```
## S3 method for class 'mfrm_plausible_values'
summary(object, digits = 3, ...)
```

**Arguments**

<code>object</code>	Output from <code>sample_mfrm_plausible_values()</code> .
<code>digits</code>	Number of digits used in numeric summaries.
<code>...</code>	Reserved for generic compatibility.

**Value**

An object of class `summary.mfrm_plausible_values` with:

- `draw_summary`: empirical summaries of the sampled values by person
- `estimates`: companion posterior EAP summaries
- `audit`: row-preparation audit
- `settings`: scoring settings
- `notes`: interpretation notes

**See Also**

[sample\\_mfrm\\_plausible\\_values\(\)](#)

**Examples**

```
toy <- load_mfrmr_data("example_core")
keep_people <- unique(toy$Person)[1:18]
toy_fit <- suppressWarnings(
  fit_mfrm(
    toy[toy$Person %in% keep_people, , drop = FALSE],
    "Person", c("Rater", "Criterion"), "Score",
    method = "MML",
    quad_points = 5,
    maxit = 15
  )
)
new_units <- data.frame(
  Person = c("NEW01", "NEW01"),
  Rater = unique(toy$Rater)[1],
  Criterion = unique(toy$Criterion)[1:2],
  Score = c(2, 3)
)
pv <- sample_mfrm_plausible_values(toy_fit, new_units, n_draws = 3, seed = 1)
summary(pv)
```

---

```
summary.mfrm_population_prediction
```

*Summarize a population-level design forecast*

---

**Description**

Summarize a population-level design forecast

**Usage**

```
## S3 method for class 'mfrm_population_prediction'
summary(object, digits = 3, ...)
```

**Arguments**

object	Output from <code>predict_mfrm_population()</code> .
digits	Number of digits used in numeric summaries.
...	Reserved for generic compatibility.

**Value**

An object of class `summary.mfrm_population_prediction` with:

- design: requested future design
- overview: run-level overview
- forecast: facet-level forecast table
- ademp: simulation-study metadata
- notes: interpretation notes

**See Also**

[predict\\_mfrm\\_population\(\)](#)

**Examples**

```
spec <- build_mfrm_sim_spec(
  n_person = 40,
  n_rater = 4,
  n_criterion = 4,
  raters_per_person = 2,
  assignment = "rotating"
)
pred <- predict_mfrm_population(
  sim_spec = spec,
  n_person = 60,
  reps = 2,
  maxit = 10,
  seed = 123
)
summary(pred)
```

---

summary.mfrm\_signal\_detection

*Summarize a DIF/bias screening simulation*

---

**Description**

Summarize a DIF/bias screening simulation

**Usage**

```
## S3 method for class 'mfrm_signal_detection'  
summary(object, digits = 3, ...)
```

**Arguments**

object	Output from <a href="#">evaluate_mfrm_signal_detection()</a> .
digits	Number of digits used in numeric summaries.
...	Reserved for generic compatibility.

**Value**

An object of class `summary.mfrm_signal_detection` with:

- overview: run-level overview
- detection\_summary: aggregated detection rates by design
- ademp: simulation-study metadata carried forward from the original object
- notes: short interpretation notes, including the bias-side screening caveat

**See Also**

[evaluate\\_mfrm\\_signal\\_detection\(\)](#), [plot.mfrm\\_signal\\_detection](#)

**Examples**

```
sig_eval <- suppressWarnings(evaluate_mfrm_signal_detection(  
  n_person = 20,  
  n_rater = 3,  
  n_criterion = 3,  
  raters_per_person = 2,  
  reps = 1,  
  maxit = 10,  
  bias_max_iter = 1,  
  seed = 123  
))  
summary(sig_eval)
```

---

summary.mfrm\_threshold\_profiles

*Summarize threshold-profile presets for visual warning logic*

---

**Description**

Summarize threshold-profile presets for visual warning logic

**Usage**

```
## S3 method for class 'mfrm_threshold_profiles'  
summary(object, digits = 3, ...)
```

**Arguments**

object	Output from <a href="#">mfrm_threshold_profiles()</a> .
digits	Number of digits used for numeric summaries.
...	Reserved for generic compatibility.

**Details**

Summarizes available warning presets and their PCA reference bands used by [build\\_visual\\_summaries\(\)](#).

**Value**

An object of class `summary.mfrm_threshold_profiles`.

**Interpreting output**

- `thresholds`: raw preset values by profile (strict, standard, lenient).
- `threshold_ranges`: per-threshold span across profiles (sensitivity to profile choice).
- `pca_reference`: literature bands used for PCA narrative labeling.

Larger Span in `threshold_ranges` indicates settings that most change warning behavior between strict and lenient modes.

**Typical workflow**

1. Inspect `summary(mfrm_threshold_profiles())`.
2. Choose profile (strict / standard / lenient) for project policy.
3. Override selected thresholds in [build\\_visual\\_summaries\(\)](#) only when justified.

**See Also**

[mfrm\\_threshold\\_profiles\(\)](#), [build\\_visual\\_summaries\(\)](#)

**Examples**

```
profiles <- mfrm_threshold_profiles()  
summary(profiles)
```

---

summary.mfrm\_unit\_prediction  
*Summarize posterior unit scoring output*

---

## Description

Summarize posterior unit scoring output

## Usage

```
## S3 method for class 'mfrm_unit_prediction'  
summary(object, digits = 3, ...)
```

## Arguments

object	Output from <a href="#">predict_mfrm_units()</a> .
digits	Number of digits used in numeric summaries.
...	Reserved for generic compatibility.

## Value

An object of class `summary.mfrm_unit_prediction` with:

- estimates: posterior summaries by person
- audit: row-preparation audit
- settings: scoring settings
- notes: interpretation notes

## See Also

[predict\\_mfrm\\_units\(\)](#)

## Examples

```
toy <- load_mfrmr_data("example_core")  
keep_people <- unique(toy$Person)[1:18]  
toy_fit <- suppressWarnings(  
  fit_mfrm(  
    toy[toy$Person %in% keep_people, , drop = FALSE],  
    "Person", c("Rater", "Criterion"), "Score",  
    method = "MML",  
    quad_points = 5,  
    maxit = 15  
  )  
)  
new_units <- data.frame(  
  Person = c("NEW01", "NEW01"),
```

```

Rater = unique(toy$Rater)[1],
Criterion = unique(toy$Criterion)[1:2],
Score = c(2, 3)
)
pred_units <- predict_mfrm_units(toy_fit, new_units)
summary(pred_units)

```

---

unexpected\_after\_bias\_table

*Build an unexpected-after-adjustment screening report*

---

### Description

Build an unexpected-after-adjustment screening report

### Usage

```

unexpected_after_bias_table(
  fit,
  bias_results,
  diagnostics = NULL,
  abs_z_min = 2,
  prob_max = 0.3,
  top_n = 100,
  rule = c("either", "both")
)

```

### Arguments

<code>fit</code>	Output from <code>fit_mfrm()</code> .
<code>bias_results</code>	Output from <code>estimate_bias()</code> .
<code>diagnostics</code>	Optional output from <code>diagnose_mfrm()</code> for baseline comparison.
<code>abs_z_min</code>	Absolute standardized-residual cutoff.
<code>prob_max</code>	Maximum observed-category probability cutoff.
<code>top_n</code>	Maximum number of rows to return.
<code>rule</code>	Flagging rule: "either" or "both".

### Details

This helper recomputes expected values and residuals after interaction adjustments from `estimate_bias()` have been introduced.

`summary(t10)` is supported through `summary()`. `plot(t10)` is dispatched through `plot()` for class `mfrm_unexpected_after_bias` (type = "scatter", "severity", "comparison").

**Value**

A named list with:

- `table`: unexpected responses after bias adjustment
- `summary`: one-row summary (includes baseline-vs-after counts)
- `thresholds`: applied thresholds
- `facets`: analyzed bias facet pair

**Interpreting output**

- `summary`: before/after unexpected counts and reduction metrics.
- `table`: residual unexpected responses after bias adjustment.
- `thresholds`: screening settings used in this comparison.

Large reductions indicate bias terms explain part of prior unexpectedness; persistent unexpected rows indicate remaining model-data mismatch.

**Typical workflow**

1. Run `unexpected_response_table()` as baseline.
2. Estimate bias via `estimate_bias()`.
3. Run `unexpected_after_bias_table(...)` and compare reductions.

**Further guidance**

For a plot-selection guide and a longer walkthrough, see [mfrmr\\_visual\\_diagnostics](#) and `vignette("mfrmr-visual-diagnostics", package = "mfrmr")`.

**Output columns**

The `table` data.frame has the same structure as `unexpected_response_table()` output, with an additional `BiasAdjustment` column showing the bias correction applied to each observation's expected value.

The `summary` data.frame contains:

**TotalObservations** Total observations analyzed.

**BaselineUnexpectedN** Unexpected count before bias adjustment.

**AfterBiasUnexpectedN** Unexpected count after adjustment.

**ReducedBy, ReducedPercent** Reduction in unexpected count.

**See Also**

[estimate\\_bias\(\)](#), [unexpected\\_response\\_table\(\)](#), [bias\\_count\\_table\(\)](#), [mfrmr\\_visual\\_diagnostics](#)

**Examples**

```
toy <- load_mfrmr_data("example_bias")
fit <- fit_mfrmr(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 25)
diag <- diagnose_mfrmr(fit, residual_pca = "none")
bias <- estimate_bias(fit, diag, facet_a = "Rater", facet_b = "Criterion", max_iter = 2)
t10 <- unexpected_after_bias_table(fit, bias, diagnostics = diag, top_n = 20)
summary(t10)
p_t10 <- plot(t10, draw = FALSE)
class(p_t10)
```

---

unexpected\_response\_table

*Build an unexpected-response screening report*

---

**Description**

Build an unexpected-response screening report

**Usage**

```
unexpected_response_table(
  fit,
  diagnostics = NULL,
  abs_z_min = 2,
  prob_max = 0.3,
  top_n = 100,
  rule = c("either", "both")
)
```

**Arguments**

fit	Output from <code>fit_mfrmr()</code> .
diagnostics	Optional output from <code>diagnose_mfrmr()</code> .
abs_z_min	Absolute standardized-residual cutoff.
prob_max	Maximum observed-category probability cutoff.
top_n	Maximum number of rows to return.
rule	Flagging rule: "either" (default) or "both".

**Details**

A response is flagged as unexpected when:

- rule = "either":  $|StdResidual| \geq abs\_z\_min$  OR  $ObsProb \leq prob\_max$
- rule = "both": both conditions must be met.

The table includes row-level observed/expected values, residuals, observed-category probability, most-likely category, and a composite severity score for sorting.

**Value**

A named list with:

- **table**: flagged response rows
- **summary**: one-row overview
- **thresholds**: applied thresholds

**Interpreting output**

- **summary**: prevalence of unexpected responses under current thresholds.
- **table**: ranked row-level diagnostics for case review.
- **thresholds**: active cutoffs and flagging rule.

Compare results across rule = "either" and rule = "both" to assess how conservative your screening should be.

**Typical workflow**

1. Start with rule = "either" for broad screening.
2. Re-run with rule = "both" for strict subset.
3. Inspect top rows and visualize with `plot_unexpected()`.

**Further guidance**

For a plot-selection guide and a longer walkthrough, see [mfrmr\\_visual\\_diagnostics](#) and `vignette("mfrmr-visual-diagnostics", package = "mfrmr")`.

**Output columns**

The table data.frame contains:

**Row** Original row index in the prepared data.

**Person** Person identifier (plus one column per facet).

**Score** Observed score category.

**Observed, Expected** Observed and model-expected score values.

**Residual, StdResidual** Raw and standardized residuals.

**ObsProb** Probability of the observed category under the model.

**MostLikely, MostLikelyProb** Most probable category and its probability.

**Severity** Composite severity index (higher = more unexpected).

**Direction** "Higher than expected" or "Lower than expected".

**FlagLowProbability, FlagLargeResidual** Logical flags for each criterion.

The summary data.frame contains:

**TotalObservations** Total observations analyzed.

**UnexpectedN, UnexpectedPercent** Count and share of flagged rows.

**AbsZThreshold, ProbThreshold** Applied cutoff values.

**Rule** "either" or "both".

**See Also**

[diagnose\\_mfrm\(\)](#), [displacement\\_table\(\)](#), [fair\\_average\\_table\(\)](#), [mfrmr\\_visual\\_diagnostics](#)

**Examples**

```
toy_full <- load_mfrm_data("example_core")
toy_people <- unique(toy_full$Person)[1:12]
toy <- toy_full[toy_full$Person %in% toy_people, , drop = FALSE]
fit <- suppressWarnings(
  fit_mfrm(toy, "Person", c("Rater", "Criterion"), "Score", method = "JML", maxit = 10)
)
t4 <- unexpected_response_table(fit, abs_z_min = 1.5, prob_max = 0.4, top_n = 5)
summary(t4)
p_t4 <- plot(t4, draw = FALSE)
class(p_t4)
```

# Index

- analyze\_dff, 10
- analyze\_dff(), 5, 6, 10, 11, 72–74, 90, 91, 125–127, 159
- analyze\_dif (analyze\_dff), 10
- analyze\_dif(), 5, 72–74, 91, 125, 159
- analyze\_facet\_equivalence, 13
- analyze\_facet\_equivalence(), 5, 164, 165
- analyze\_residual\_pca, 16
- analyze\_residual\_pca(), 4, 5, 70, 178, 179
- anchor\_to\_baseline, 19
- anchor\_to\_baseline(), 39, 40, 66, 67, 126, 127
- apa\_table, 22
- apa\_table(), 5, 128, 129, 135, 139, 140, 216, 217
- as.data.frame.mfrm\_fit, 24, 93
- audit\_mfrm\_anchors, 25
- audit\_mfrm\_anchors(), 5, 64, 119, 120, 126, 141, 217, 218
  
- bias\_count\_table, 28
- bias\_count\_table(), 237
- bias\_interaction\_report, 30
- bias\_interaction\_report(), 5, 33, 35, 41, 123, 155, 156, 221
- bias\_iteration\_report, 32
- bias\_pairwise\_report, 33
- build\_apa\_outputs, 35
- build\_apa\_outputs(), 4–6, 23, 41, 51, 74, 82, 95, 113, 122, 123, 128–131, 135, 201, 204, 219
- build\_equating\_chain, 37
- build\_equating\_chain(), 21, 66, 67, 126, 153, 154
- build\_fixed\_reports, 40
- build\_fixed\_reports(), 29, 32, 33, 35, 82, 104, 123, 135, 198
- build\_mfrm\_manifest, 42
- build\_mfrm\_manifest(), 5, 45, 46, 95, 96
- build\_mfrm\_replay\_script, 44
- build\_mfrm\_replay\_script(), 5, 43, 44, 95, 96
- build\_mfrm\_sim\_spec, 47
- build\_mfrm\_sim\_spec(), 5, 6, 85, 89, 98, 135, 186, 188, 211, 212
- build\_visual\_summaries, 49
- build\_visual\_summaries(), 4–6, 37, 70, 95, 128, 129, 135, 138, 175, 177, 201, 207, 234
  
- category\_curves\_report, 52
- category\_curves\_report(), 54, 101, 123, 130, 131
- category\_structure\_report, 53
- category\_structure\_report(), 53, 101, 123, 130, 131
- compare\_mfrm, 55
- compare\_mfrm(), 13
- compute\_information, 58
- compute\_information(), 169–171
  
- data\_quality\_report, 60
- data\_quality\_report(), 84, 130, 131, 201, 214, 216
- describe\_mfrm\_data, 62
- describe\_mfrm\_data(), 5, 27, 61, 122, 130, 144, 145, 223, 224
- detect\_anchor\_drift, 64
- detect\_anchor\_drift(), 20, 21, 39, 40, 126, 127, 153, 154
- diagnose\_mfrm, 67, 92, 93, 157, 158
- diagnose\_mfrm(), 4–6, 10, 14, 17, 18, 20–23, 31, 32, 34, 35, 42, 44, 45, 50, 54, 58, 65, 71, 74, 75, 77–80, 85, 89, 94, 98–102, 104, 106–109, 113, 115, 116, 120, 122, 123, 126–129, 131, 134, 135, 137, 155, 160, 162, 164, 166, 168, 171, 174, 178–181, 183, 184, 186, 187, 192–194, 197, 198, 200, 201, 203, 204, 206, 207, 213,

- 215, 216, 225, 226, 230, 236, 238, 240
- dif\_interaction\_table, 70
- dif\_interaction\_table(), 5, 13, 73, 74, 125, 159
- dif\_report, 73
- dif\_report(), 5, 13, 72, 127, 159
- displacement\_table, 74
- displacement\_table(), 109, 160–162, 240
- ej2021\_combined (ej2021\_data), 76
- ej2021\_combined\_intercal (ej2021\_data), 76
- ej2021\_data, 76, 117, 118
- ej2021\_study1 (ej2021\_data), 76
- ej2021\_study1\_intercal (ej2021\_data), 76
- ej2021\_study2 (ej2021\_data), 76
- ej2021\_study2\_intercal (ej2021\_data), 76
- estimate\_all\_bias, 78
- estimate\_all\_bias(), 5
- estimate\_bias, 80
- estimate\_bias(), 4–6, 13, 22, 28, 29, 31–35, 37, 40–42, 45, 72, 78, 79, 90, 91, 94, 102, 104–106, 113, 125, 134, 135, 137, 155, 156, 197, 200, 206, 207, 220, 221, 236, 237
- estimation\_iteration\_report, 83
- estimation\_iteration\_report(), 130, 131, 203, 204, 214
- evaluate\_mfrm\_design, 84
- evaluate\_mfrm\_design(), 5, 6, 91, 135, 146, 147, 187, 188, 195, 196, 213, 224, 225
- evaluate\_mfrm\_signal\_detection, 88
- evaluate\_mfrm\_signal\_detection(), 5, 151, 233
- export\_mfrm, 25, 92
- export\_mfrm(), 43, 95, 96
- export\_mfrm\_bundle, 93
- export\_mfrm\_bundle(), 5, 43–46, 101, 123
- extract\_mfrm\_sim\_spec, 96
- extract\_mfrm\_sim\_spec(), 5, 49, 85, 89, 135, 186–188, 211, 212
- facet\_quality\_dashboard, 104
- facet\_quality\_dashboard(), 5, 79, 94, 95, 166, 167, 228
- facet\_statistics\_report, 106
- facet\_statistics\_report(), 128–131, 186
- facets\_chisq\_table, 98
- facets\_chisq\_table(), 16, 116, 130, 162, 164
- facets\_output\_file\_bundle, 100
- facets\_output\_file\_bundle(), 123
- facets\_parity\_report, 102
- facets\_parity\_report(), 123, 198
- fair\_average\_table, 108
- fair\_average\_table(), 16, 75, 168, 169, 203, 204, 222, 240
- fit\_mfrm, 24, 25, 92, 93, 110, 157, 158
- fit\_mfrm(), 4–7, 9, 10, 13, 14, 16, 17, 21, 23, 27, 28, 31, 32, 34, 35, 42, 44–46, 50, 52–55, 57, 58, 60, 61, 64, 68, 70–72, 74, 77–80, 83–85, 89, 94, 97, 98, 100, 102, 104, 107, 108, 115, 118–120, 122, 123, 126–129, 131, 134, 135, 137, 145, 149, 150, 155, 160, 162, 164, 166, 168, 171, 174, 178, 180, 183, 184, 186, 187, 189, 191–193, 197, 198, 200, 201, 203, 204, 206, 208, 213–215, 218, 224, 229, 230, 236, 238
- graphics::image(), 159
- interrater\_agreement\_table, 114
- interrater\_agreement\_table(), 99, 171, 173
- list\_mfrmr\_data, 117
- list\_mfrmr\_data(), 77, 118
- load\_mfrmr\_data, 118
- load\_mfrmr\_data(), 5, 76, 77, 117, 125
- make\_anchor\_table, 119
- make\_anchor\_table(), 5, 20, 21, 27, 40, 44, 65, 67, 95, 126, 127, 141
- measurable\_summary\_table, 120
- measurable\_summary\_table(), 21, 130, 194, 216
- mfrm\_threshold\_profiles, 137
- mfrm\_threshold\_profiles(), 51, 207, 234
- mfrmr (mfrmr-package), 4
- mfrmr-package, 4, 112
- mfrmr\_compatibility\_layer, 5, 41, 61, 84, 101, 104, 122, 131, 137, 204, 214
- mfrmr\_example\_bias (mfrmr\_example\_data), 124

- mfrmr\_example\_core
  - (mfrmr\_example\_data), 124
- mfrmr\_example\_data, 124
- mfrmr\_linking\_and\_dff, 5, 13, 21, 67, 123, 126, 131, 134, 137, 216
- mfrmr\_reporting\_and\_apa, 5, 23, 37, 70, 113, 123, 128, 131, 134, 137
- mfrmr\_reports\_and\_tables, 5, 41, 53, 54, 61, 84, 101, 107, 122, 123, 127, 129, 130, 134, 137, 214
- mfrmr\_visual\_diagnostics, 5, 18, 29, 53, 54, 70, 72, 116, 121–123, 127, 129, 131, 132, 137, 148, 150, 154, 161, 162, 169, 173, 177, 182, 184, 193, 194, 204, 216, 237, 239, 240
- mfrmr\_workflow\_methods, 5, 113, 123, 127, 129, 131, 134, 135, 148, 204, 228
- mfrmRFacets (run\_mfrm\_facets), 202
- mfrmRFacets(), 5, 123, 135, 137
- plot(), 217
- plot.apa\_table, 139
- plot.apa\_table(), 217
- plot.mfrm\_anchor\_audit, 140
- plot.mfrm\_bundle, 142
- plot.mfrm\_data\_description, 144
- plot.mfrm\_design\_evaluation, 87, 146, 196, 225
- plot.mfrm\_design\_evaluation(), 196
- plot.mfrm\_facets\_run, 147
- plot.mfrm\_fit, 148
- plot.mfrm\_fit(), 5, 53, 54, 134, 137, 147, 148, 184, 194, 204
- plot.mfrm\_signal\_detection, 151, 233
- plot\_anchor\_drift, 152
- plot\_anchor\_drift(), 40, 67, 126, 127, 130, 132, 134
- plot\_bias\_interaction, 154
- plot\_bias\_interaction(), 5, 31, 32, 82, 133, 134, 136, 142, 221
- plot\_bubble, 157
- plot\_bubble(), 150, 154
- plot\_dif\_heatmap, 159
- plot\_dif\_heatmap(), 5, 13, 72, 74, 125–127, 154
- plot\_displacement, 160
- plot\_displacement(), 75, 132–134, 136, 142, 143, 156, 169, 175, 182
- plot\_facet\_equivalence, 164
- plot\_facet\_equivalence(), 5, 15, 16
- plot\_facet\_quality\_dashboard, 166
- plot\_facet\_quality\_dashboard(), 5, 228
- plot\_facets\_chisq, 162
- plot\_facets\_chisq(), 99, 107, 132, 134, 142, 173, 175
- plot\_fair\_average, 158, 167
- plot\_fair\_average(), 109, 142, 143, 162, 175, 182
- plot\_information, 169
- plot\_information(), 60, 184
- plot\_interrater\_agreement, 171
- plot\_interrater\_agreement(), 116, 132–134, 142, 164, 175
- plot\_qc\_dashboard, 173
- plot\_qc\_dashboard(), 106, 133–136, 147, 148, 162, 164, 169, 173, 177, 182, 204, 207
- plot\_qc\_pipeline, 176
- plot\_qc\_pipeline(), 207
- plot\_residual\_pca, 178
- plot\_residual\_pca(), 5, 18, 132, 134, 142
- plot\_unexpected, 158, 180
- plot\_unexpected(), 132–134, 136, 142, 143, 162, 169, 175, 239
- plot\_wright\_unified, 182
- plot\_wright\_unified(), 132, 135, 150
- precision\_audit\_report, 184
- precision\_audit\_report(), 60, 128–131
- predict\_mfrm\_population, 186
- predict\_mfrm\_population(), 5, 6, 42, 45, 94, 95, 135, 189, 191, 232
- predict\_mfrm\_units, 189
- predict\_mfrm\_units(), 5, 6, 42, 45, 94, 95, 135, 209, 210, 235
- print.mfrm\_anchor\_drift
  - (detect\_anchor\_drift), 64
- print.mfrm\_anchored\_fit
  - (anchor\_to\_baseline), 19
- print.mfrm\_apa\_text, 191
- print.mfrm\_equating\_chain
  - (build\_equating\_chain), 37
- print.summary.mfrm\_anchor\_drift
  - (detect\_anchor\_drift), 64
- print.summary.mfrm\_anchored\_fit
  - (anchor\_to\_baseline), 19
- print.summary.mfrm\_equating\_chain
  - (build\_equating\_chain), 37

- psych::describe(), 63
- rating\_scale\_table, 192
- rating\_scale\_table(), 53, 54, 122, 130, 131, 204
- recommend\_mfrm\_design, 195
- reference\_case\_audit, 196
- reference\_case\_audit(), 123, 135
- reference\_case\_benchmark, 198
- reference\_case\_benchmark(), 123, 135
- reporting\_checklist, 200
- reporting\_checklist(), 5, 23, 37, 44, 79, 95, 96, 122, 123, 128–131, 134, 135, 186, 204
- run\_mfrm\_facets, 202
- run\_mfrm\_facets(), 5, 42, 44–46, 94, 123, 135, 137, 147, 148, 227, 228
- run\_qc\_pipeline, 205
- run\_qc\_pipeline(), 176, 177
- sample\_mfrm\_plausible\_values, 208
- sample\_mfrm\_plausible\_values(), 5, 6, 42, 45, 94, 95, 135, 230, 231
- simulate\_mfrm\_data, 210
- simulate\_mfrm\_data(), 5, 48, 49, 85, 87, 90, 91, 98
- specifications\_report, 213
- specifications\_report(), 61, 84, 130, 131, 201
- stats::optim(), 112
- stats::p.adjust(), 11, 71
- subset\_connectivity\_report, 215
- subset\_connectivity\_report(), 13, 126, 127, 130, 131, 134, 135
- summary(), 140, 219
- summary.apa\_table, 216
- summary.mfrm\_anchor\_audit, 217
- summary.mfrm\_anchor\_drift
  - (detect\_anchor\_drift), 64
- summary.mfrm\_anchored\_fit
  - (anchor\_to\_baseline), 19
- summary.mfrm\_apa\_outputs, 218
- summary.mfrm\_bias, 220
- summary.mfrm\_bundle, 221
- summary.mfrm\_data\_description, 223
- summary.mfrm\_design\_evaluation, 87, 147, 196, 224
- summary.mfrm\_design\_evaluation(), 195, 196
- summary.mfrm\_diagnostics, 225
- summary.mfrm\_equating\_chain
  - (build\_equating\_chain), 37
- summary.mfrm\_facet\_dashboard, 228
- summary.mfrm\_facet\_dashboard(), 167
- summary.mfrm\_facets\_run, 227
- summary.mfrm\_fit, 229
- summary.mfrm\_fit(), 5, 107, 137, 224, 226, 228
- summary.mfrm\_plausible\_values, 210, 230
- summary.mfrm\_population\_prediction, 188, 231
- summary.mfrm\_signal\_detection, 151, 232
- summary.mfrm\_threshold\_profiles, 233
- summary.mfrm\_unit\_prediction, 191, 235
- unexpected\_after\_bias\_table, 236
- unexpected\_after\_bias\_table(), 29
- unexpected\_response\_table, 238
- unexpected\_response\_table(), 75, 101, 109, 180–182, 222, 237
- utils::zip(), 94