



Item response theory with mirtverse :: CHEAT SHEET

mirtverse sits on top of **mirt**. mirt estimates the models; mirtverse helps you choose a model, read the results, make figures and share them. Every function takes an ordinary mirt model, so you can mix the two packages freely.

Install

```
install.packages("mirtverse", repos = c(
  "https://okanbulut.r-universe.dev",
  "https://cloud.r-project.org"))
library(mirtverse) # loads mirt too
```

Choose a model

start here

```
data <- expand.table(LSAT7)
sel <- select_model(data)
mod <- sel$best # an ordinary mirt model

select_model(data, candidates, dims = 1, criterion = "BIC")
  Fit and recommend a model: Rasch, 2PL or 3PL for 0/1 items;
  PCM, GPCM or graded otherwise.

autoplot(sel) · interpret(sel) · tidy(sel)
  Compare the candidates as a plot, in words or as a table.

mirt(data, 1, itemtype = "2PL")
  Or fit the model yourself with mirt.
```

Read the results in plain language

```
interpret(mod, alpha = 0.01)
  Fit, reliability, weak or misfitting items and local dependence,
  written as sentences. Also works on select_model(), dif()
  and fit_rt() results.

interpret_prompt(mod, audience)
  Pack the results into a prompt to paste into an AI assistant for a
  longer narrative. Nothing is sent anywhere.
```

Tidy outputs

data frames for dplyr, gt ...

```
tidy(mod, IRTpars = TRUE, conf.int = FALSE)
  One row per item parameter, in the IRT (a, b, c) metric.

glance(mod)
  One row of model fit: logLik, AIC, BIC, M2, RMSEA, CFI,
  reliability.

augment(mod, method = "EAP", person_fit = FALSE)
  The data with each person's score and standard error added.

item_table(mod, fit_stats = "S_X2")
  One row per item: parameters, fit and a status flag.

tidy() also works on DIF, response-time, bank and pipeline objects.
```

Differential item functioning

```
d <- dif(data, group = gender, model = "2PL")
tidy(d); interpret(d); gg_dif(d)

dif(x, group, model, items, which.par, p_adjust = "BH")
  LR tests, effect sizes, adjusted p-values.
```

Graphics

every plot is a ggplot

 gg_trace(mod) item curves; type = "expected"	 gg_info(mod) type = "test", "items", "se"
 gg_wright(mod) persons and items on one scale	 gg_itemfit(mod) item fit with a cutoff
 gg_q3(mod, cutoff = 0.2) local dependence	 gg_scores(mod) distribution of person scores
 gg_empirical(mod, item) model curve vs. observed	 gg_dif(d) type = "effect" or "curves"

```
autoplot(mod, type)
  Every plot from one function: "trace", "expected", "info",
  "iteminfo", "se", "wright", "itemfit", "q3", "scores".

gg_trace(mod) + theme_mirtverse(base_size = 11)
  Change labels, colours and themes as with any ggplot.

Common arguments: which.items picks items, theta_lim = c(-4, 4) sets the ability range.
```

Response times

```
sim <- sim_rt(N = 1500, J = 20, rapid = 0.08)
mod <- mirt(sim$responses, 1, "2PL")
fit <- fit_rt(mod, sim$rt)

fit_rt(x, rt, flag_method = "residual")
  Joint model for responses and response times (van der Linden,
  2007) with person speed and rapid-guessing flags. Also
  "nt10", "nt15", "nt20".

tidy(fit) · glance(fit) · augment(fit) · autoplot(fit) · interpret(fit)
  Item time parameters, fit, person speed and flags, plots and
  plain-language findings.

sim_rt(N, J, rho = -0.3, sd_speed, rapid, seed)
  Simulate responses and times, with a share of rapid guessers.
```

Coming from another IRT package?

```
mirtify(code, run = FALSE, notes = TRUE)
  Rewrite ltm, eRm, TAM or irtQ code as mirt code, with notes on
  what differs. Give it a string or an .R file.

mirtify_crosswalk(package)
  Table of equivalent functions.
```

Share the results

```
item_card(mod, item) · item_cards(mod, file)
  A one-page health card per item with a traffic-light status, for
  review meetings.

mv_report(sel, file = "report.html", dif, cards, code)
  One HTML report: model comparison, interpretation, tables,
  graphics, a sum-score table, item cards and the R code to
  reproduce it.

mv_dashboard(mod, title, panels, dif, rt)
  A read-only Shiny dashboard for people who do not use R.

export_dashboard(mod, "results.html", format = "html")
  One offline HTML file to email, or format = "shiny" for
  shinyapps.io or Posit Connect.
```

Click, or describe it in words

```
launch_workbench(data)
  Point-and-click app that writes the R code for every step.

mv_ask_setup("claude") # once
scr <- mv_ask("Fit a 2PL, check DIF by gender
  and make a report", data = mydata)
run_script(scr)
save_script(scr, "analysis.R")

mv_ask(request, data, anonymize = FALSE, dry_run = FALSE)
  Turn a request into a tested script. dry_run = TRUE shows what
  would be sent; anonymize = TRUE hides column names. Works
  offline with a keyword parser.

profile_data(data) · mv_plan() · update_plan(plan, ...)
  Inspect the data, or write and edit the analysis plan yourself.
```

Teaching labs

```
run_lab("model-choice") · run_lab("decisions")
  Interactive tutorials: students fit live models and make testing
  decisions, with feedback.

list_labs() · lab_truth()
  Available labs; the planted answers, for instructors.

certexam · certexam_key
  A simulated 32-item certification exam in English and French,
  with planted problems.
```



The assessment pipeline :: CHEAT SHEET

One object carries a test from blueprint to bank monitoring. Each stage does its work, then runs a **gate**: checks that record **flags** for you to consider before the next stage. Every decision is logged, and `p$code` keeps the R code for the whole run.

```
p <- mv_pipeline("Certification exam")
p <- add_responses(p, certexam, items = 3:34,
  key = certexam_key, group = "language")
p <- calibrate(p, model = "2PL")
flags(p)
pipeline_report(p, "report.html")
```

0	1	2	3	4	5	6	7	8	9	10	11	12
Blueprint	Item generation	Item review	Field-test design	Data intake	Calibration	Linking and bank	Forms	Cut scores	Scoring	Test security	Reporting	Bank monitoring

Gates and flags

decide, then move on

info for awareness
review look before you continue
block fix, drop or override with a reason
flags(p, status = "open", level, stage)
 List flags; status = "all" includes decided ones.
acknowledge(p, check, target, note)
 Accept a flag, with an optional note.
override(p, check, target, note)
 Go past a flag; a written reason is required.
drop_items(p, items, note)
 Remove items and mark their flags fixed.

Rules and strictness

mv_pipeline(title, rules = mv_rules(), strict = FALSE)
 Gates warn by default. With strict = TRUE, open review and block flags stop the next stage until a decision is recorded.
mv_rules(..., preset = "standards")
 Every threshold, changeable by name: `mv_rules(q3 = 0.25, reliability = 0.85)`. preset = "lenient" relaxes them for pilot studies.
print(p) shows each stage's status; **tidy(p)** lists all flags as a data frame.

Build the test

stages 0 to 3

0 set_blueprint(p, cells, construct, use, reliability)
 Content cells and item counts, e.g. `c(Ethics = 4, Law = 3)`.
1 generate_items(p, n, type, provider) **LLM**
 Draft MC or open-ended items; **verify_items()** checks keys; **add_items(p, items)** brings your own.
2 review_packet(p, file, reviewers)
 Write a review form; **read_reviews(p, reviews)** reads it back.
3 design_field_test(p, n_forms, n_per_form)
 Plan forms and anchors, then simulate parameter recovery.

Calibrate and assemble

stages 4 to 7

4 add_responses(p, data, items, key, group, id, rt)
 Score raw answers with the key. Checks missing data, miskeys, straight-lining, rapid guessing and sparse categories.
5 calibrate(p, model = "auto", dif, reference)
 Fit the model. Checks convergence, model and item fit, local dependence, parameter ranges, DIF, reliability and person fit.
6 link_to_bank(p, bank, method)
 Put the new items on the bank's scale: "stocking-lord", "haebara", "mean-sigma", "mean-mean" or "fixed".
bank_items(p, name) stores them; **mv_bank(mod)** starts a bank from a model.
7 assemble_forms(p, length, n_forms, theta, content, blueprint, enemies)
 Build parallel forms that meet the blueprint and keep enemy items apart.
simulate_cat(p, n, min_sem, max_items, criteria = "MI")
 Simulate a computerized adaptive test from the bank.
assemble_mst(p, router_length, module_length, n_modules)
 Build and simulate a multistage test.

The certexam example

```
p <- drop_items(p, "Q14", note = "Miskey")
p <- calibrate(p, model = "2PL")
p <- override(p, check = "dif_large",
  note = "French version revised")
p <- bank_items(p, name = "Certification bank")
p <- set_cuts(p, cuts = 18, scale = "raw",
  labels = c("Fail", "Pass"))
p <- score_persons(p)
```

The full walk-through is in the *Assessment pipeline* article on the website.

Decide, score and report

stages 8 to 12

8 set_cuts(p, cuts, scale = "theta", labels, panel)
 Set cut scores on the theta or raw scale, with classification accuracy and consistency. **item_map(p, rp = 0.67)** maps items to the scale for standard setting.
9 score_persons(p, method = "EAP")
 Person scores with standard errors, checked for reliability and extreme scores. **score_open(p, responses)** scores open-ended answers with a rubric and an LLM or your own trained model. **LLM**
10 check_security(p, leaked, window, site)
 Preknowledge, leaked items, drift, copying and unusual timing.
11 pipeline_report(p, file)
 An HTML technical report with every stage, flag and decision, and the R code.
12 monitor_bank(p, window, date, content)
 After each administration: item exposure, bank depth per content cell, reliability trend and unused items. **retire_items(p, items, status, note)** takes items out of use.

What each stage needs

ellmer **LLM**
 Item generation, AI review screening and open-ended scoring, with your own API key (Claude, OpenAI or a local Ollama model).
eatATA
 Form assembly.
mirtCAT
 CAT and multistage simulation.
 These are suggested packages: install them only for the stages you use. Linking and cut-score methods are built in.

Prefer clicking?

The workbench has a **Pipeline** tab that runs the same stages, shows the flags and writes the code: `launch_workbench()`.