Package 'xpose.xtras'

September 1, 2026

Title Extra Functionality for the 'xpose' Package

Version 0.2.2

Description Adding some at-present missing functionality, or functions unlikely to be added to the base 'xpose' package. This includes some diagnostic plots that have been missing in translation from 'xpose4', but also some useful features that truly extend the capabilities of what can be done with 'xpose'. These extensions include the concept of a set of 'xpose' objects, and diagnostics for likelihood-based models.

License MIT + file LICENSE

Encoding UTF-8

Imports checkmate, cli, colorspace, dplyr (>= 1.1.2), forcats (>= 1.0.0), GGally, ggplot2 (>= 3.4.2), glue, grDevices, grid, lifecycle, magrittr, pmxcv, purrr (>= 1.0.1), readr (>= 2.1.4), rlang, stats, stringr (>= 1.5.0), tibble (>= 3.2.1), tidyr (>= 1.3.0), tidyselect, utils, vctrs, xpose

Suggests bbr, callr, conflicted, DiagrammeR, knitr, nlmixr2, nlmixr2data, nlmixr2est, pkgload, rmarkdown, rxode2, spelling, testthat (>= 3.0.0), vdiffr, xpose.nlmixr2

Config/testthat/edition 3

Depends R (>= 3.5)

LazyData true

VignetteBuilder knitr

URL https://jprybylski.github.io/xpose.xtras/, https://github.com/jprybylski/xpose.xtras

Language en-US

Config/Needs/website rmarkdown

Config/roxygen2/version 8.0.0

Additional_repositories https://mpn.metworx.com/snapshots/stable/2026-06-25

NeedsCompilation no

Author John Prybylski [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0001-5802-0539>)

Maintainer John Prybylski <jprybylski@gmail.com>

Repository CRAN

Date/Publication 2026-09-01 10:10:03 UTC

Contents

`add_cov_association` *Describe parameter/covariate associations*

Description

The relationship between a structural parameter and a covariate can be described, so that the covariate's effect on that parameter – and the uncertainty of that effect – can later be visualized with `xplot_forest()` (via `prm_cov()`/`prm_contcov()`/`prm_catcov()`).

This is deliberately parallel to `add_prm_association()`: the same formula-based declaration style, the same two-stage check-then-process validation, and the same "redeclare to replace" upsert behavior. It is a separate mechanism (own storage, own getters) because a covariate association needs more shape than an omega association – a covariate column, a *required* reference value, and (for categorical covariates or `custom()`) more than one theta – and it produces a *range* of effect sizes rather than a single scalar CV.

Usage

```
add_cov_association(xpdb, ..., .problem, .subprob, .method, quiet)

drop_cov_association(xpdb, ..., .problem, .subprob, .method, quiet)
```

Arguments

xpdb	<xp_xtras> object
...	<dynamic-dots> One or more formulas that define associations between a parameter and a covariate. One list of formulas can also be used, but a warning is generated. For drop_cov_association, these should be formulas of the form param ~ covariate (both bare, unquoted selectors; covariate must be the literal covariate column name as declared).
.problem	<numeric> Problem number to apply this relationship.
.subprob	<numeric> Subprob number to apply this relationship.
.method	<numeric> Method to apply this relationship.
quiet	Silence extra output.

Details

Format for associations is:

LHS ~ fun(COVARIATE, THETA..., ref = ..., ...)

- LHS: Selector for a fixed-effect (theta) parameter, exactly as in add_prm_association() (the{m}, {name} or {label}, unquoted). Multiple parameters can share one association with + (eg, a covariate that affects both CL and Q through the same theta).
- RHS COVARIATE: The first (positional) argument. The bare, unquoted name of a contcov/catcov column (see xpose::xp_var()) – *not* a fixed-effect or omega selector.
- RHS THETA...: One or more further positional arguments, selecting the fixed-effect parameter(s) that carry the covariate effect magnitude (same selector rules as LHS – the{m}/{name}/{label}, unquoted). How many are expected depends on fun; see the built-in list below.
- RHS ref: **Required, named, no default**, for every association regardless of fun. This is the covariate value (for continuous covariates) or raw level (for categorical covariates) that the effect is normalized against – the point at which the reported effect ratio is exactly 1. There is no way to safely infer this from the xpdb alone (NONMEM control streams commonly normalize a covariate against a hardcoded constant baked into the code, which is invisible to xpose), so it must always be stated explicitly, the same way add_prm_association()'s nmboxcox requires an explicit lambda.

All built-ins express the covariate's effect as a multiplicative effect_ratio on the parameter's typical value, and are constructed so that effect_ratio == 1 whenever the covariate equals ref, *regardless of the theta value*. Available built-ins:

- linear(COV, THETA, ref=): $1 + \theta(COV - ref)$
- power(COV, THETA, ref=): $(COV/ref)^{\theta}$ (allometric)

- `exponential(COV, THETA, ref=)`: $e^{\theta(COV-ref)}$
- `additive(COV, THETA, ref=)`: $(\theta+COV)/(\theta+ref)$. An uncommon but simple form where the covariate is added directly (with an implicit coefficient of 1, unlike `linear`'s explicit slope) to an intercept-like `THETA`, eg `CL = THETA(n) + WT` in the underlying model.
- `hockey(COV, THETA_LO, THETA_HI, ref=, brk=ref)`: PsN's "hockey-stick" two-slope piecewise-linear model – $1 + \theta_{lo}(COV - ref)$ when `COV <= brk`, $1 + \theta_{hi}(COV - ref)$ when `COV > brk`. `brk` (the breakpoint) defaults to `ref` (PsN's usual default: breakpoint = normalization reference), but can be given separately, eg for a covariate normalized to its observed median while the clinically meaningful cutpoint is a round number (`ref = 90, brk = 60` for an eGFR-like covariate).
- `catshift(COV, THETA..., ref=)`: One theta per non-reference raw level of a categorical covariate, `effect_ratio = 1 + THETA_i` for level `i` (1 at `ref`). Assumes the typical NONMEM pattern of one theta per non-reference category (eg `IF (RACE.EQ.2) CLCOV = THETA(9)`). Thetas are matched to non-reference levels in ascending raw-value order – if that order is ambiguous or wrong for a given model, use `custom()` instead.

For anything else, `custom(COV, THETA..., ref=, fun=)` is the escape hatch: `fun` is a function of `(cov, ref, theta)` (`theta` is always a numeric vector, even when only one `THETA` selector is given) returning the effect ratio. Because `custom()` can't be verified by construction the way the built-ins can, `add_cov_association()` validates it at declaration time by evaluating `fun(ref, ref, theta)` for a few probe values of `theta` (not the currently-fitted value, which could coincidentally pass while `fun` is still wrong for other theta values) and requires each to equal `1`; if it doesn't, the error states the required invariant and shows what `fun` actually returned, rather than failing silently or only much later during plotting.

A note on parameter/theta scale:

The computed effect ratio never touches the LHS parameter's own fitted value – only the covariate-effect theta(s), the covariate value, and `ref` – so it does not matter whether the LHS was itself parameterized on a log, logit, or identity scale in the control stream (`get_prm()`'s `transform` argument only affects diagonal OMEGA/SIGMA reporting (variance -> SD, covariance -> correlation); THETA values are always the raw fitted estimate regardless of `transform`, so there is nothing to reconcile there either).

What *is* assumed is that the chosen association – a builtin or `custom()` – is an exact match for the functional form actually used in the model code, including any scale factor baked into that code (eg a covariate effect entered as `THETA(n)*(COV-ref)/100`). There is no way to verify this from the `xpdb` alone; if a builtin's literal formula (see above) doesn't match the real model, the result will be a numerically valid but silently wrong effect ratio, not an error – the same caveat `add_prm_association()` already carries for CV% calculation.

If the covariate-effect theta itself is not already on the scale a builtin expects (eg it was fitted on a logit or other transformed scale, or needs some other rescaling to match one of the literal formulas above), transform it back with `mutate_prm()` *before* declaring the association – the same recommended workflow as `add_prm_association()`'s own untransformed-theta requirement.

Value

An updated `xp_xtras` object

See Also

add_prm_association(), prm_cov(), mutate_prm()

Examples

```
# xpdb_x's THETA7 ("CRCL on CL") is a genuine covariate effect already
# in the model, so this is a faithful (if allometric-flavored, for
# illustration) description of it:
xpdb_x %>%
  add_cov_association(TVCL ~ power(CLCR, THETA7, ref = 64)) %>%
  prm_cov()

# hockey-stick (PsN-style): different slope above/below the reference
xpdb_x %>%
  add_cov_association(TVCL ~ hockey(CLCR, THETA7, THETA4, ref = 64)) %>%
  prm_cov()

# Categorical: one theta per non-reference level. SEX has 2 levels
# (1, 2), so catshift needs exactly one theta for the non-reference
# level; THETA4 is reused here purely for illustration.
xpdb_x %>%
  add_cov_association(TVCL ~ catshift(SEX, THETA4, ref = 1)) %>%
  prm_cov()

# custom(): fun(cov, ref, theta) must equal 1 when cov == ref
xpdb_x %>%
  add_cov_association(
    TVCL ~ custom(CLCR, THETA7, ref = 64,
                  fun = function(cov, ref, theta) (cov/ref)^theta)
  ) %>%
  prm_cov()

# Dropping an association is easy
bad_assoc <- xpdb_x %>%
  add_cov_association(TVCL ~ power(CLCR, THETA7, ref = 64))
bad_assoc %>%
  drop_cov_association(TVCL ~ CLCR) %>%
  prm_cov()
```

---

add_prm_association *Describe parameter associations*

Description

The relationship between structural parameters and omega parameters can be described. This is useful if it deviates from the typical log-normal.

Default transformations are those that are built into pmxcv, but see examples for how associations can be described for other relationships.

Note: When these associations are used to calculate CV%, it is assumed that the value for the theta parameter is *untransformed*. So, if a parameter is fitted in the logit scale, the value should be transformed back to normal scale with `mutate_prm()` (eg, `mutate_prm(the~plogis)` before declaring `the~logit(ome)`.

Usage

```
add_prm_association(xpdb, ..., .problem, .subprob, .method, quiet)

drop_prm_association(xpdb, ..., .problem, .subprob, .method, quiet)
```

Arguments

`xpdb`	<xp_xtras> object
`...`	... <dynamic-dots> One or more formulas that define associations between parameters. One list of formulas can also be used, but a warning is generated. For `drop_prm_association`, these dots should be selectors for which associations will be dropped (`the2, the3,...`). Fixed effect selectors only will work.
`.problem`	<numeric> Problem number to apply this relationship.
`.subprob`	<numeric> Problem number to apply this relationship.
`.method`	<numeric> Problem number to apply this relationship.
`quiet`	Silence extra output.

Details

At time of writing, the built-in distributions for `pmxcv` are below. Those marked with an asterisk require a fixed effect parameter to calculate CV.

- `log` typical log-normal. Optional `exact` parameter (if `TRUE`, default, will not calculate with integration); this is unrelated to the `cvtype` option. **Note**, if `cvtype` option is set to `"sqrt"`, log-normal `get_prm` CVs will use the square root, not any integration or analytical estimate, regardless of how this association is specified.
- `logexp`* modified log-normal `log(1+X)`
- `logit`* logit-normal
- `arcsin`* arcsine-transform
- `nmboxcox`* Box-Cox transform as typically implemented in pharmacometrics. Requires a `lambda` parameter.

To pass a custom parameter, use `custom` transform, and pass `pdist` and `qdist` to that transform. See Examples.

Reminder about `qdist` and `pdist`: Consider that `qlogis` transforms a proportion to a continuous, unbounded number; it is the `logit` transform. The `plogis` function converts a continuous, unbounded number to a proportion; it is the *inverse* `logit` transform. Other R `stats` functions work similarly, and as such functions used as `qdist` and `pdist` values are expected to act similarly.

Note that the functions used in describing associations are not real functions, it is just the syntax for this application. Based on examples, be mindful of where positional arguments would acceptable

and where named arguments are required. Care has been given to provide a modest amount of flexibility with informative errors for fragile points, but not every error can be anticipated. If this function or downstream results from it seem wrong, the association syntax should be scrutinized. These "functions" are not processed like in mutate_prm, so (eg) the2 will not be substituted for the value of the2; if lambda is a fitted value (like the2), in that edge case the value of the2 should be written explicitly in the association formula, and if any mutate_prm changes the2 then users should be mindful of the new association needed. This may be updated in the future.

Format for associations is: LHS~fun(OMEGA, args...)

- LHS: Selector for a fixed effect parameter. Can be the{m} (eg, the1), {name} (eg, THETA1) or {label} (eg, TVCL). These should *not* be quoted. Multiple associations can be defined at once with +. Cannot be empty.
- RHS: Should be a simple call to only one function, which should be custom or one of the built-in distributions or custom(...). A lot of things can look like simple calls, so may not break immediately; keep to the described format and everything should be fine.
- RHS OMEGA: Selector for omega variable. Similar rules to the fixed effect selector. Can be ome{m}, {name} or {label}, limited to diagonal elements. Should *not* be quoted. OMEGA is not a named argument (OMEGA={selector} should **not** be considered valid); whatever is used as the first argument to the "function" will be considered an OMEGA selector. **NOTE**, if selecting an OMEGA parameter by name (eg, OMEGA(2,2)), backticks (eg `OMEGA(2,2)`) must be used or else the selection will throw an error.
- RHS args: Applies when the distribution has extra arguments. If these are limited to 1, can be passed by position (eg, lambda for nmboxcox and exact for log). For custom(), qdist, pdist and any arguments needed to pass to them should be named.

For the nmboxcox transformation, a lambda value (especially negative ones) may not work well with the integration-based CV estimation. This may occur even if the lambda is fitted and stable in that fitting, but it cannot be predicted which ones will be affected. This note is intended to forewarn that this might happen.

Value

An updated xp_xtras object

References

Prybylski, J.P. Reporting Coefficient of Variation for Logit, Box-Cox and Other Non-log-normal Parameters. Clin Pharmacokinet 63, 133-135 (2024). doi:10.1007/s40262023013432

See Also

dist.intcv

Examples

```
pheno_base %>%
   add_prm_association(the1~log(IIVCL),V~log(IIVV)) %>%
   get_prm() # get_prm is the only way to see the effect of associations
```

```
# These values are not fitted as logit-normal, but
# just to illustrate:
pheno_final %>%
   add_prm_association(the1~logit(IIVCL),Vpkg~logit(IIVV)) %>%
   get_prm()

# ... same for Box-Cox
pheno_base %>%
   add_prm_association(V~nmboxcox(IIVV, lambda=0.5)) %>%
   # Naming the argument is optional
   add_prm_association(CL~nmboxcox(IIVCL, -0.1)) %>%
   get_prm()

# A 'custom' use-case is when logexp, log(1+X), is
# desired but 1 is too large.
# Again, for this example, treating this like it applies here.
pheno_base %>%
  add_prm_association(V~custom(IIVV, qdist=function(x) log(0.001+x),
        pdist=function(x) exp(x)-0.001)) %>%
   get_prm()

# Dropping association is easy
bad_assoc <- pheno_final %>%
   add_prm_association(the1~logit(IIVCL),Vpkg~logit(IIVV))
bad_assoc %>% get_prm()
bad_assoc %>%
  drop_prm_association(the1) %>%
  get_prm()
```

`add_process_preset` *Add, apply, list, amend or remove* `xpdb` *processing presets*

Description

[Experimental]

A "process preset" is a one-sided formula (using `.x` for the incoming `xpdb`, as in a `purrr`-style lambda) or a plain function that bundles up a repeated processing pipeline – e.g. converting to `xp_xtras`, dropping unused `ETA`s, assigning labels/levels – so it can be re-applied with `process_preset()` instead of being retyped for every model.

`add_process_preset()` stores a preset in the current session (by name, or an auto-incrementing integer if `name` isn't given). `process_preset()` applies a stored preset to an `xpdb`. `print_process_preset()` lists stored presets. `remove_process_preset()` deletes one. `amend_process_preset()` replaces an existing preset's definition in place.

Usage

```
add_process_preset(
  preset,
```

```
  name = NULL,
  overwrite = FALSE,
  persist = FALSE,
  ask = TRUE,
  profile = "project"
)

process_preset(xpdb, preset, ...)

print_process_preset(name = NULL)

remove_process_preset(name, persist = FALSE, ask = TRUE, profile = "project")

amend_process_preset(
  name,
  preset,
  persist = FALSE,
  ask = TRUE,
  profile = "project"
)
```

Arguments

preset	<formula> or <function> One-sided formula (e.g. `~ .x %>% as_xpdb_x() %>% set_var_type(na = any_of(paste0("ETA", 5:9)))`) or a function taking an xpdb as its first argument. For `process_preset()`, instead the `character` name or `numeric` (1-based) index of a previously-added preset.
name	<character(1)> Name to store/look up/amend the preset under. For `add_process_preset()`, defaults to the next unused integer (as a string) if omitted.
overwrite	<logical(1)> If a preset already exists under `name`, should it be replaced? (default: `FALSE`, i.e. error)
persist	<logical(1)> Write the resulting set of presets out to a `.Rprofile` so they're available in future sessions too? See Details. (default: `FALSE`)
ask	<logical(1)> When `persist = TRUE`, ask for interactive confirmation before writing? (default: `TRUE`; only ever consulted when `rlang::is_interactive()` is already `TRUE`)
profile	<character(1)> Where to persist to when `persist = TRUE`: `"project"` (default, `.Rprofile` in `getwd()`), `"user"` (the user-level profile), or a literal file path.
xpdb	<xpose_data> or <xp_xtras> object to apply the preset to
...	For `process_preset()`, forwarded to the preset if it is a function (ignored by formula presets, which only ever see `xpdb`)

Value

`add_process_preset()`/`amend_process_preset()`/`remove_process_preset()` return the (`character(1)`) preset name, invisibly. `process_preset()` returns the processed `xpdb`. `print_process_preset()` returns the printed presets (a named list), invisibly.

Persistence and CRAN policy

By default, presets are session-only: they vanish when R restarts. Set `persist = TRUE` to additionally write the current set of presets to a `.Rprofile` so they're recreated automatically in future sessions.

Per CRAN policy, a package must not write to files outside `tempdir()` without the user's explicit, interactive consent, and never as a side effect of a non-interactive process (`R CMD check`, tests, vignette builds, `Rscript`, ...). Accordingly, `persist = TRUE`:

- only ever writes when `rlang::is_interactive()` is `TRUE` – it errors otherwise, so it is always a no-op under `R CMD check`/`testthat`/`knitr` – and
- asks for confirmation (via `utils::askYesNo()`) before writing, unless `ask = FALSE` is passed explicitly by the (already-interactive) caller.

The target file defaults to a **project**-scoped `.Rprofile` (in `getwd()`), which only affects R sessions started in that directory; pass `profile = "user"` to instead target the user-level profile (`Sys.getenv("R_PROFILE_USER")`, falling back to `~/.Rprofile`), or any string to use it as a literal file path. Presets are written as a single marked block (bounded by `# >>> xpose.xtras process presets ... >>>` / `# <<< ... <<<`) so re-syncing replaces the whole block rather than accumulating duplicate calls, and the block is removed entirely once the last preset is deleted. Persisted presets must be self-contained – since they're recreated from deparsed source on each new session, they cannot depend on transient local variables from the session that created them.

Formula presets and `.x` vs `.`

Use `.x` (not a bare `.`) as the placeholder for the incoming xpdb in a formula preset, e.g. `~ .x %>% as_xpdb_x() %>% set_var_type(...)`. A *leading* `.` immediately before `%>%` is itself magrittr syntax for building a reusable function (see `?magrittr::`%>%``): `~ . %>% f()` would silently return a function instead of applying it, since the formula's own `.` placeholder collides with magrittr's.

See Also

xpose::xpose_data

Examples

```
add_process_preset(
  ~ .x %>% as_xpdb_x() %>% set_var_types(na = any_of(paste0("ETA", 5:9))),
  name = "drop_higher_etas"
)
print_process_preset()

xpdb_ex_pk_processed <- xpose::xpdb_ex_pk %>%
  process_preset("drop_higher_etas")

amend_process_preset(
  "drop_higher_etas",
  ~ .x %>% as_xpdb_x() %>% set_var_types(na = any_of(paste0("ETA", 7:9)))
)

remove_process_preset("drop_higher_etas")
```

add_relationship *Add relationship(s) to an xpose_set*

Description

Add relationship(s) to an xpose_set

Usage

```
add_relationship(xpdb_s, ..., .warn = TRUE, .remove = FALSE)

remove_relationship(xpdb_s, ...)
```

Arguments

xpdb_s	<xpose_set> An xpose_set object
...	<dynamic-dots> One or more formulas that define relationships between models. One list of formulas can also be used, but a warning is generated.
.warn	<logical> Should warnings be generated for non-formula inputs? (default: TRUE)
.remove	<logical> Should listed relationships be removed? (default: FALSE)

Value

An xpose_set object with relationships added

Examples

```
xpdb_set %>%
  add_relationship(mod1~fix2) # ouroboros

xpdb_set %>%
  remove_relationship(fix1~mod2) # split down the middle
```

add_watermark *Add a watermark to a plot*

Description

Overlays large, semi-transparent, rotated text across a ggplot or xpose_plot object (e.g. "DRAFT", "PRELIMINARY", "CONFIDENTIAL"). Calling it directly always adds a watermark (falling back to the built-in defaults below if nothing else is configured). It is also applied automatically by print.xpose_plot() and ggsave_xp() whenever a default_watermark is actually configured (see the auto_apply entry in set_xtras_options()) – unlike this function, that automatic trigger never invents an unconfigured watermark on its own.

label/colour/alpha/size/angle/fontface resolve with the same precedence as apply_default_labs(): (in increasing precedence) the xpose.xtras.default_watermark R option, xpdb-level defaults set via set_default_watermark() (when xpdb is supplied), then the argument itself when explicitly passed. Built-in fallbacks ("DRAFT", "grey50", 0.3, 24, 30, "bold") apply if a setting isn't resolved from any of those.

Usage

```
add_watermark(plot, label, colour, alpha, size, angle, fontface, xpdb = NULL)
```

Arguments

plot	<ggplot> or <xpose_plot> object
label	<character> Watermark text
colour	<character> Text colour, passed to grDevices::adjustcolor()
alpha	<numeric> Transparency of the watermark text, between 0 (invisible) and 1 (opaque)
size	<numeric> Font size in points
angle	<numeric> Rotation angle in degrees (counter-clockwise)
fontface	<character> Font face, passed to grid::gpar()
xpdb	<xpose_data> or <xp_xtras> object plot was built from, used to resolve xpdb-level defaults (see set_default_watermark())

Value

plot, with the watermark layer added

See Also

set_xtras_options() for the full list of xpose.xtras.* session options, and get_xtras_option() to check which tier (option/xpdb) is currently dominant for a given xpdb.

Examples

```
p <- xpose::dv_vs_ipred(xpose::xpdb_ex_pk)
add_watermark(p, label = "PRELIMINARY")

options(xpose.xtras.default_watermark = list(label = "CONFIDENTIAL", colour = "red"))
add_watermark(p)
options(xpose.xtras.default_watermark = NULL)
```

add_xpdb *Add one or more* xpdb *objects to an* xpose_set

Description

Add one or more xpdb objects to an xpose_set

Usage

```
add_xpdb(xpdb_s, ..., .relationships = NULL)
```

Arguments

xpdb_s	<xpose_set> An xpose_set object
...	<dynamic-dots> One or more xpdb objects to add to the set
.relationships	<list> A list of relationships between the xpdb objects.

Value

An xpose_set object with the new xpdb objects added

Examples

```
data("xpdb_ex_pk", package = "xpose")

add_xpdb(xpdb_set, ttt=xpdb_ex_pk)
```

apply_default_labs *Apply default label overrides to a plot*

Description

Resolves title/subtitle/caption/tag labels for plot from (in increasing precedence):

1. the xpose.xtras.default_labs R option (a named list, see examples),
2. defaults set on xpdb via set_default_labs(), when xpdb is supplied (a rendered plot does not retain enough of its source xpdb to look this up automatically – pass it explicitly),
3. values passed directly via ....

By default only labels not already set on the plot are filled in; set overwrite = TRUE to replace existing labels too. print.xpose_plot() calls this automatically whenever xpose.xtras.auto_apply is enabled (the default; see set_xtras_options()), but apply_default_labs() itself is a standalone function that doesn't depend on that print method existing, so it works the same regardless of whether that method survives issue #36's eventual removal.

Usage

```
apply_default_labs(plot, ..., xpdb = NULL, overwrite = FALSE)
```

Arguments

plot	<ggplot> or <xpose_plot> object
...	<dynamic-dots> Direct overrides for title/subtitle/caption/tag, taking precedence over both the option- and xpdb-level defaults
xpdb	<xpose_data> or <xp_xtras> object plot was built from, used to resolve xpdb-level defaults (see set_default_labs()) and to resolve @keyword placeholders in any of the labels. If omitted and plot is an xpose_plot, @keyword placeholders are still resolved (using the reduced context xpose attaches to the plot) but xpdb-level defaults are not available
overwrite	<logical> Replace labels already present on plot (default FALSE, meaning only missing labels are filled in)

Value

plot, with resolved labels applied

See Also

set_xtras_options() for the full list of xpose.xtras.* session options, and get_xtras_option() to check which tier (option/xpdb) is currently dominant for a given xpdb.

Examples

```
options(xpose.xtras.default_labs = list(caption = "Draft -- do not distribute"))
p <- xpose::dv_vs_ipred(xpose::xpdb_ex_pk)
apply_default_labs(p)
options(xpose.xtras.default_labs = NULL)
```

---

as_leveler *Level-defining helper functions*

Description

as_leveler() is the generic constructor behind every leveler: it tags a character vector x so that set_var_levels() recognizes it as a set of levels rather than a plain formula list. x[1] is mapped to the raw data value .start_index, x[2] to .start_index + 1, and so on. lvl_bin(), lvl_sex(), and lvl_inord() are thin wrappers around as_leveler() for common cases (binary Yes/No, Male/Female sex, and pre-ordered factors, respectively); call as_leveler() directly when defining your own levels, eg more than two categories, a custom starting index, or an unordered custom label set that the built-in wrappers don't cover.

Usage

```
as_leveler(x, .start_index = 1, .ordered = FALSE)

is_leveler(x)

lvl_bin(x = c("No", "Yes"), .start_index = 0)

lvl_sex()

lvl_inord(x, .start_index = 1, .ordered = TRUE)
```

Arguments

x	<character> vector of levels
.start_index	<numeric> starting index for levels
.ordered	<logical> should these levels be treated as an ordered factor (see base::factor) wherever they're consumed (eg val2lvl())?

Value

Special character vector suitable to be used as leveler

Examples

```
# Roll your own leveler for a case the convenience wrappers don't cover,
# eg an unordered, 3-category custom covariate coded 0/1/2 in the data
arm_levels <- as_leveler(c("Placebo", "Low dose", "High dose"), .start_index = 0)
arm_levels
is_leveler(arm_levels)

# The convenience wrappers below are just as_leveler() under the hood, eg
# lvl_bin() is equivalent to as_leveler(c("No", "Yes"), .start_index = 0)
set_var_levels(xpdb_x,
  SEX = lvl_sex(),
  MED1 = lvl_bin(),
  MED2 = lvl_inord(c("n","y"), .start_index = 0)
  )
```

---

as_xpdb_x *Convert an object to an* xpose_data *and* xp_xtras *object*

Description

This function masks the default in xpose package, adding the xp_xtras class to default xpose_data objects.

Usage

```
as_xpdb_x(x)

as_xp_xtras(x)

check_xpdb_x(x, .warn = TRUE)

check_xp_xtras(...)
```

Arguments

x	Suspected xp_xtras object
.warn	<logical> Whether to warn if xpose_data but not xp_xtras
...	Forwarded

Value

<xpose_data> and <xp_xtras> object

Examples

```
xp_x <- as_xpdb_x(xpose::xpdb_ex_pk)
check_xpdb_x(xp_x)
```

---

attach_nlmixr2 *Attach nlmixr2 fit object to xpose data object*

Description

Attach nlmixr2 fit object to xpose data object

Usage

```
attach_nlmixr2(xpdb, obj)
```

Arguments

xpdb	<xpose_data> The object upon which to attach the fit
obj	<nlmixr2FitData> Result of the nlmixr2 fit

Value

An object of the same class as xpdb with an additional element.

Examples

```
## Not run:
# Based on an example from nlmixr2 documentation
xpdb_nlmixr2 <- nlmixr_example("xpdb_nlmixr2")

one.cmt <- function() {
  ini({
    tka <- 0.45 # Ka
    tcl <- log(c(0, 2.7, 100)) # Log Cl
    tv <- 3.45; label("log V")
    eta.ka ~ 0.6
    eta.cl ~ 0.3
    eta.v ~ 0.1
    add.sd <- 0.7
  })
  model({
    ka <- exp(tka + eta.ka)
    cl <- exp(tcl + eta.cl)
    v <- exp(tv + eta.v)
    linCmt() ~ add(add.sd)
  })
}

theo_sd_fit <- nlmixr2::nlmixr2(one.cmt, nlmixr2data::theo_sd,
    "focei", control = nlmixr2::foceiControl(print = 0))

attach_nlmixr2(
  xpdb_nlmixr2, theo_sd_fit
) %>%
  as_xpdb_x() %>%
  print() # fit will be mentioned in print() method

## End(Not run)
```

---

backfill_iofv *Add individual objective function to data*

Description

Add individual objective function to data

Usage

```
backfill_iofv(xpdb, .problem = NULL, .subprob = NULL, .label = "iOFV")
```

Arguments

xpdb	<xpose_data> or <xp_xtras> object
.problem	Problem number

.subprob	Subproblem number
.label	The name of the new column. iOFV is the default.

Details

This function will only work for objects with software listed as nonmem or nlmixr2. For nonmem, the object should haves a phi file and with an OBJ column in that file. For nlmixr2, the fit object data should have individual observation likelihoods in a column called NLMIXRLLIKOBS (this is a current standard, but is checked at runtime).

Value

<xp_xtras> object with new column in the data and a column with iofv var type.

Examples

```
xpdb_x %>%
  backfill_iofv() %>%
  list_vars()
```

---

backfill_nlmixr2_props

Populate some properties from nlmixr2 fit

Description

Populate some properties from nlmixr2 fit

Usage

```
backfill_nlmixr2_props(xpdb)
```

Arguments

xpdb	<xpose_data> object

Details

This function will currently backfill:

- condn
- nsig

Examples

```
## Not run:
xpdb_nlmixr2 <- nlmixr_example("xpdb_nlmixr2")

xpdb_nlmixr2 %>%
  set_prop(condn = "not implemented") %>%
  get_prop("condn")

xpdb_nlmixr2 %>%
  set_prop(condn = "not implemented") %>%
  backfill_nlmixr2_props() %>%
  get_prop("condn")

## End(Not run)
```

`catdv_vs_dvprobs` *Non-simulation based likelihood model diagnostic*

Description

These plots attempt to provide a means of verifying that the estimated likelihoods and probabilities for categorical outcomes are captured within the model.

When the smooth spline is included (`type` includes `"s"`), it is expected that the overall trend is up and to the right; a relatively flat trend suggests that the modeled likelihood is inconsistent with the observed outcome.

Usage

```
catdv_vs_dvprobs(
  xpdb,
  mapping = NULL,
  cutpoint = 1,
  type = "vbs",
  title = "@y vs. @x | @run",
  subtitle = "Ofv: @ofv, Number of individuals: @nind",
  caption = "@dir",
  tag = NULL,
  xlab = c("probability", "basic"),
  facets,
  .problem,
  quiet,
  ...
)
```

Arguments

xpdb	<xp_xtras> or <xpose_data> object
mapping	ggplot2 style mapping
cutpoint	<numeric> Of defined probabilities, which one to use in plots.
type	See Details.
title	Plot title
subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag
xlab	Either use the typical basic x-axis label (the cutpoint-defined column name) or label it based on the probability/likelihood it is estimating.
facets	Additional facets
.problem	Problem number
quiet	Silence extra debugging output
...	Any additional aesthetics.

Details

For type-based customization of plots:

- b box-whisker (using default quantiles)
- p points (from geom_dotplot)
- v violin (from geom_violin)
- o outliers (show outliers)
- l line through 0 (or as indicated in hline_yintercept or yline_xintercept)
- s smooth line (from geom_smooth)
- j jitter points (from geom_jitter)
- c connecting lines for jitter points (from geom_path)

Value

The desired plot

Examples

```
# Test M3 model
pkpd_m3 %>%
  # Need to ensure var types are set
  set_var_types(catdv=BLQ,dvprobs=LIKE) %>%
  # Set probs
  set_dv_probs(1, 1~LIKE, .dv_var = BLQ) %>%
  # Optional, but useful to set levels
  set_var_levels(1, BLQ = lvl_bin()) %>%
  # Plot with basic xlab makes no assumptions
```

```
  catdv_vs_dvprobs(xlab = "basic")

# Test categorical model
vismo_xpdb <- vismo_pomod  %>%
  set_var_types(.problem=1, catdv=DV, dvprobs=matches("^P\\d+$")) %>%
  set_dv_probs(.problem=1, 0~P0,1~P1,ge(2)~P23)

# Various cutpoints (note axes labels and texts)
vismo_xpdb %>%
  catdv_vs_dvprobs(xlab = "basic")
vismo_xpdb %>%
  catdv_vs_dvprobs(cutpoint = 2, xlab = "basic")
vismo_xpdb %>%
  catdv_vs_dvprobs(cutpoint = 3, xlab = "basic")

# Latter is arguably clearer with default xlab
vismo_xpdb %>%
  catdv_vs_dvprobs(cutpoint = 3)
```

`catdv_vs_ipred` *Binned calibration plot for categorical DVs*

Description

A binned alternative to `catdv_vs_dvprobs()`. The probability column associated with `cutpoint` is split into `bins` equally-sized groups, from lowest to highest predicted probability, and for each bin the observed proportion of the categorical DV meeting the cutpoint condition is calculated (i.e. the m/M observations in that bin with the target value).

For a well-specified model, the mean predicted probability of a bin should be close to the bin's observed proportion, so plotted points are expected to fall around the unity (`y = x`) line.

Usage

```
catdv_vs_ipred(
  xpdb,
  mapping = NULL,
  cutpoint = 1,
  bins = 10,
  type = "pl",
  guide = TRUE,
  title = "Observed frequency vs. predicted probability | @run",
  subtitle = "Ofv: @ofv, Number of individuals: @nind",
  caption = "@dir",
  tag = NULL,
  xlab = c("probability", "basic"),
  facets,
  .problem,
```

```
  quiet,
  ...
)
```

Arguments

xpdb	<xp_xtras> or <xpose_data> object
mapping	ggplot2 style mapping
cutpoint	<numeric> Of defined probabilities, which one to use in plots.
bins	<numeric> Number of (roughly) equally-sized bins used to group the probability column, from lowest to highest.
type	String setting the type of plot to be used: line l, point p, smooth s and text t, or any combination thereof. See xpose::xplot_scatter().
guide	Include the unity (y = x) guide line?
title	Plot title
subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag
xlab	Either use the typical basic x-axis label (the cutpoint-defined column name) or label it based on the probability/likelihood it is estimating.
facets	Additional facets
.problem	Problem number
quiet	Silence extra debugging output
...	Any additional aesthetics.

Value

The desired plot

See Also

catdv_vs_dvprobs()

Examples

```
# Test M3 model
pkpd_m3 %>%
  # Need to ensure var types are set
  set_var_types(catdv=BLQ,dvprobs=LIKE) %>%
  # Set probs
  set_dv_probs(1, 1~LIKE, .dv_var = BLQ) %>%
  # Optional, but useful to set levels
  set_var_levels(1, BLQ = lvl_bin()) %>%
  # Plot with 5 bins
  catdv_vs_ipred(bins = 5)
```

```
# Test categorical model
vismo_xpdb <- vismo_pomod  %>%
  set_var_types(.problem=1, catdv=DV, dvprobs=matches("^P\\d+$")) %>%
  set_dv_probs(.problem=1, 0~P0,1~P1,ge(2)~P23)

# Various cutpoints and bin counts
vismo_xpdb %>%
  catdv_vs_ipred(bins = 8, xlab = "basic")
vismo_xpdb %>%
  catdv_vs_ipred(cutpoint = 2, bins = 8, xlab = "basic")
vismo_xpdb %>%
  catdv_vs_ipred(cutpoint = 3, bins = 8, xlab = "basic")
```

`catdv_vs_occ` *Longitudinal binned observed vs. predicted plot for categorical DVs*

Description

A longitudinal alternative to `catdv_vs_ipred()` and to xpose's own `xpose::dv_preds_vs_idv()` for categorical outcomes. Rather than binning by predicted probability (as `catdv_vs_ipred()` does) or plotting raw per-subject values against a continuous independent variable, this bins observations by a discrete, typically ordered grouping variable (eg an `occ`-typed occasion column) and plots the observed proportion meeting the cutpoint condition alongside the mean predicted probability, one point/line per bin.

Usage

```
catdv_vs_occ(
  xpdb,
  mapping = NULL,
  bin = NULL,
  cutpoint = 1,
  type = "pl",
  title = "Observed and predicted probability vs. @x | @run",
  subtitle = "Ofv: @ofv, Number of individuals: @nind",
  caption = "@dir",
  tag = NULL,
  facets,
  .problem,
  quiet,
  ...
)
```

Arguments

`xpdb`	<xp_xtras> or <xpose_data> object
`mapping`	ggplot2 style mapping

bin	<tidyselect> Column to bin/group by. Defaults to the first occ-typed column (see set_var_types()). If that column has defined levels (see set_var_levels()), those labels (and their order) are used; otherwise raw values are coerced to a factor as-is.
cutpoint	<numeric> Of defined probabilities, which one to use in plots.
type	String setting the type of plot to be used: point p, line l, and smooth s, or any combination thereof. See xplot_binned().
title	Plot title
subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag
facets	Additional facets
.problem	Problem number
quiet	Silence extra debugging output
...	Any additional aesthetics.

Value

The desired plot

See Also

catdv_vs_ipred(), catdv_vs_dvprobs()

Examples

```
# Derive an occasion column (TIME is in hours here) and level it in
# visit order
vismo_xpdb <- vismo_pomod %>%
  set_var_types(.problem = 1, catdv = DV, dvprobs = matches("^P\\d+$")) %>%
  set_dv_probs(.problem = 1, 0~P0, 1~P1, ge(2)~P23) %>%
  xpose::mutate(OCC = ceiling((TIME + 1) / 24), .problem = 1) %>%
  set_var_types(.problem = 1, occ = OCC) %>%
  set_var_levels(.problem = 1, OCC = lvl_inord(paste("Day", 1:12)))

vismo_xpdb %>%
  catdv_vs_occ()

vismo_xpdb %>%
  catdv_vs_occ(cutpoint = 3)
```

check_levels *Verify validity of level list*

Description

Verify validity of level list

Usage

```
check_levels(lvl_list, index, .ordered = character())
```

Arguments

lvl_list	<list> of formulas or leveler functions
index	Index of xp_xtras object
.ordered	<character> Names of columns to be forced to an ordered factor, as passed to set_var_levels()

Value

Nothing, warning or error

check_xpose_set *Check an* xpose_set *object*

Description

Check an xpose_set object

Usage

```
check_xpose_set(xpdb_s, .warn = TRUE)

check_xpose_set_item(xpdb_s_i, .example = xpdb_set)
```

Arguments

xpdb_s	<xpose_set> An xpose_set object
.warn	<logical> Display a warning on failure.
xpdb_s_i	<xpose_set_item> An xpose_set_item object (element of an xpose_set)
.example	<xpose_set> Basis of comparison for xpose_s_i

Value

TRUE or error thrown

Examples

```
check_xpose_set(xpdb_set)

check_xpose_set_item(xpdb_set$mod1)
```

`cormat` *Parameter correlation/covariance matrix heatmap*

Description

Visualizes the parameter correlation (or covariance) matrix as a heatmap, filling a gap left behind in translation from `xpose4`. Values come from `get_cov_matrix()`, which has built-in support for `nonmem` and `nlmixr2` models; rendering is done with the generic `xplot_heatmap()` template.

Usage

```
cormat(
  xpdb,
  type = c("correlation", "covariance"),
  .problem = NULL,
  .subprob = NULL,
  .method = NULL,
  drop_fixed = TRUE,
  digits,
  title,
  subtitle = "Ofv: @ofv, Condition number: @condn",
  caption = "@dir",
  tag = NULL,
  quiet,
  ...
)
```

Arguments

`xpdb`	<xp_xtras> or <xpose_data> object
`type`	<character> Either `"correlation"` (default) or `"covariance"`
`.problem`	<numeric> Problem number to use. Uses the xpose default if not provided.
`.subprob`	<numeric> Subproblem number to use. Uses the xpose default if not provided.
`.method`	<character> Method to use. Uses the xpose default if not provided.
`drop_fixed`	<logical> Passed to `get_cov_matrix()`
`digits`	Number of significant digits to display in cell labels. Defaults to `reportable_digits()`
`title`	Plot title
`subtitle`	Plot subtitle

caption	Plot caption
tag	Plot tag
quiet	Silence extra debugging output
...	Additional aesthetics, passed to xplot_heatmap()

Details

Only the upper triangle of the matrix is drawn, as it is symmetric. Fixed-effect (theta) and random-effect (omega/sigma) parameters are both included for nonmem models, when available and not fixed. For nlmixr2 models, only fixed effects are included, as nlmixr2 does not report uncertainty for random effects. See get_cov_matrix() for further details on availability; if the covariance step was not run, or did not complete successfully, an informative error is raised.

Value

The desired plot

Examples

```
cormat(xpdb_x)
cormat(xpdb_x, type = "covariance")
```

---

cov_forest *Covariate effect forest plot*

Description

Visualizes the effect of covariates on structural parameters, as declared with add_cov_association(), as a forest plot: one row per (parameter, covariate, evaluation point), the point estimate and interval as a ratio to the parameter's typical value, with a reference line at 1.

This is the covariate-specific wrapper: it calls prm_cov() to compute the effect-size table and xplot_forest() (a generic, forest-plot-agnostic renderer, see its own documentation) to draw it.

Usage

```
cov_forest(
  xpdb,
  ...,
  type = "pilr",
  region = NULL,
  show_ref = TRUE,
  log = TRUE,
  forest_opts = list(),
  title = "Covariate effects on model parameters | @run",
  subtitle = "Ratio to typical parameter value; reference line at 1",
  caption = "@dir",
```

```
  tag = NULL,
  .problem = NULL,
  .subprob = NULL,
  .method = NULL,
  quiet
)
```

Arguments

xpdb	<xp_xtras> object with covariate associations declared via add_cov_association()
...	<dynamic-dots> Forwarded to prm_cov() – eg param ~ covariate selectors, ci_method, probs, level, nsim.
type	Passed to xplot_forest(); defaults to 'pilr' (point + interval + reference line + shaded reference region – xplot_forest()'s own defaults omit the line and region, since those are cov_forest()- specific opinions, not generic ones). Including "v" adds a violin/density layer of the raw simulation draws behind each interval; this forces prm_cov(keep_draws = TRUE), which in turn requires ci_method = "simulation" (the default) – pass ci_method = "delta" in ... together with type containing "v" and it will error, since no draws exist for the delta method.
region	<numeric(2)> c(low, high) bounds for the shaded reference region (type includes "r", the default); NULL (default) falls back to c(0.8, 1.25), a common bioequivalence-style "no relevant effect" band.
show_ref	<logical> Include the reference row(s) (effect/ ci_low/ci_high always 1, by construction, for every reference covariate value/level)? Defaults to TRUE; set FALSE to drop them from the plot – they carry no information beyond what the reference line already shows, and cutting them can reduce clutter when there are many covariates.
log	<logical> Log-scale the effect-ratio (x) axis? Defaults to TRUE. Unlike most of the package's log arguments (eg eta_vs_contcov()'s), this is a plain boolean rather than an "x"/"y"/NULL axis-selector string – cov_forest()'s orientation isn't user-configurable, so the axis being logged is never ambiguous.
forest_opts	<list> Extra named arguments forwarded to xplot_forest() (eg theme overrides), the same way pairs_opts works for cov_grid()/eta_grid(). Rarely needed since the most common override, type, is already its own argument.
title	Plot title
subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag
.problem	<numeric> Problem number
.subprob	<numeric> Subprob number
.method	<numeric> Method
quiet	Silence extra output

Value

The desired plot

See Also

add_cov_association(), prm_cov(), xplot_forest()

Examples

```
xpdb_x %>%
  add_cov_association(
    TVCL ~ power(CLCR, THETA7, ref = 64),
    TVCL ~ catshift(SEX, THETA4, ref = 1)
  ) %>%
  cov_forest()
```

---

derive_prm *Derive full parameter set for mammillary PK model*

Description

This function applies rxode2::rxDerived to model parameters.

Usage

```
derive_prm(
  xpdb,
  .prm = NULL,
  .problem,
  quiet = xpdb$options$quiet,
  prefix = ""
)

backfill_derived(
  xpdb,
  .prm = NULL,
  .problem,
  quiet = xpdb$options$quiet,
  ...,
  group_vars = "id"
)
```

Arguments

xpdb	xpdb <xpose_data> object
.prm	<tidyselect> Parameters to convert (if NULL, the function will use xp_var param types)
.problem	Optional. Problem to use.
quiet	Optional. Extra output.
prefix	If desired, apply prefix to new parameters.
...	Passed to derive_prm()
group_vars	Variable type(s) to join derived parameters on.

Value

<data.frame> of data with new parameters

Examples

```
## Not run:
nlmixr2_m3 <- nlmixr_example("nlmixr2_m3")

nlmixr2_m3 %>%
  backfill_derived() %>%
  list_vars()

derive_prm(nlmixr2_m3)

# If param has no vars, .prm should be set
pheno_base %>%
  backfill_derived(
    .prm = c(CL,V)
  ) %>%
  list_vars()

## End(Not run)
```

---

derive_shk *Derive per-individual contribution to eta shrinkage*

Description

Computes, for each selected eta, a diagnostic column highlighting each individual's contribution to shrinkage: $\log((\eta_i - \bar{\eta})^2)$, ie the log of the squared deviation from the population mean eta. Since shrinkage itself is `100 * (1 - SD(eta)/omega)` (see `recalc_shk()`), and `SD(eta)^2` is the mean of these per-individual squared deviations, this highlights which individuals are pulling shrinkage down. The `log` spreads out values close to `0`, ie individuals contributing the least (the most heavily shrunk).

derive_shk() returns the augmented data as a plain data frame, like xpose::get_data()'s output. backfill_shk() joins the new column(s) back into xpdb and tags them with the shk variable type. This has to be backfilled rather than parsed, since it isn't something NONMEM (or any other supported software) reports directly.

Usage

```
derive_shk(xpdb, ..., .problem = NULL, quiet)

backfill_shk(xpdb, ..., .problem = NULL, quiet)
```

Arguments

xpdb	<xpose_dataxpose::xpose_data> or xp_xtras object
...	<tidyselect> Which eta column(s) to derive a shrinkage contribution for. Defaults to every eta column for .problem.
.problem	<numeric> Problem number to use. Uses the xpose default if not provided.
quiet	<logical> Silence extra debugging output

Value

For derive_shk(), a data frame with one new column per selected eta, named <eta>_SHK. For backfill_shk(), the updated xp_xtras object, with those columns joined in and typed shk.

Examples

```
derive_shk(xpdb_x) %>%
  dplyr::select(ID, dplyr::ends_with("_SHK")) %>%
  head()

xpdb_x %>%
  backfill_shk() %>%
  list_vars()
```

---

desc_from_comments *Backfill utility for descriptions*

Description

A slightly more generic approach to getting model descriptions.

Usage

```
desc_from_comments(
  xpdb,
  start_check = ".*description",
  maxlines = 5,
  remove = paste0(start_check, ":\\s*"),
  extra_proc = c,
  collapse = " "
)
```

Arguments

xpdb	<xpose_data> or <xp_xtras> object
start_check	Regular expression used to mark start of description. This is tested case-insensitively.
maxlines	If the number of lines after description to the first code block is more than 1, this allows a limit.
remove	By default, the start check and a colon, with optional whitespace. A regex.
extra_proc	Any extra processing that might be desired prior to collapsing the description lines. This should be a vectorized function.
collapse	Character to use when collapsing multiple lines.

Value

The description-updated <xpose_data>) object

See Also

set_prop()

Examples

```
# This has a description, but it's not visible by default
pheno_base

# It can be added with the following
pheno_base %>%
  desc_from_comments()

# Extra processing for preference can also implemented
pheno_base %>%
  desc_from_comments(extra_proc = tolower)

# If a run label ($PROB) would make a good description, use the
# following instead:
pkpd_m3 %>%
  set_prop(descr=get_prop(pkpd_m3,"label"))
```

`diagnose_constants` *Check for potential parameterization issues*

Description

This function can help diagnose potential flip-flop or other issues related to the parameterization of the model.

Usage

```
diagnose_constants(
  xpdb,
  df = NULL,
  micro_pattern = "^K(\\d+|EL?)$",
  vol_pattern = "^V(C|D|1|2|)$",
  fo_abs = "KA",
  fo_rates = c("alpha_beta", "lambda", "custom"),
  checks = list(flip_flop = NULL, neg_microvol = NULL, units_match = NULL),
  df_units = NULL,
  .problem,
  quiet = xpdb$options$quiet
)
```

Arguments

`xpdb`	<xpose_data> object
`df`	Optional <data.frame> of parameter values.
`micro_pattern`	Regex. Pattern for microconstants
`vol_pattern`	Regex. Pattern for volume parameter (should only match 1)
`fo_abs`	First-order absorption parameter (singular, fixed, not regex).
`fo_rates`	Derived ("macro") exponential rate constants (fixed). See Details
`checks`	See Details
`df_units`	Named list of units. If `NULL`, either ignore (`df`) or pull from `xpdb` object.
`.problem`	Used in fetching parameters.
`quiet`	Should parameter fetching produce output?

Details

The function prints output directly, not as an object.

A finding from these checks does not necessarily prove the parameterization is erroneous (indeed, flip-flop PK can exist), but coupled with other findings would help in diagnosing issues.

For `fo_rates`, `"alpha_beta"` and `"lambda"` are convenience placeholders meaning literally `c("ALPHA","BETA","GAMMA")` and `paste0("LAMBA",1:3)`, respectively. If capitalization or competing names will be an issue, specify a custom set of names (provide a character vector of names, do not pass `"custom"` to the

argument). If only a subset of alpha_beta or lambda are available, but these are the parameterizations used (eg, only ALPHA) these options can still be used. If LAMBDA is used alone, it will not match the "lambda" default. If naming conventions are incompatible, it is suggested xpdb or df be subject to mutation or renaming to use this function.

The available checks at this time are:

- flip_flop Checks if fo_abs are slower than the derived fo_rates.
- neg_microvol Checks if any microconstant or volume is negative. Note this check applies to parameterization of microconstants, so only a single volume (parameterizations with multiple volumes do not use microconstants) should match vol_pattern.
- units_match For any checks, verifies units are consistent. This check requires units are defined by set_var_units() or df_units for parameters applicable to a requested check.

Checks must be requested as a named list of these elements, either TRUE or FALSE (truth determines if the test is done). If the default NULL is used, test will be run if the required parameters are present.

Value

Nothing

See Also

backfill_derived()

Examples

```
## Not run:
nlmixr2_m3 <- nlmixr_example("nlmixr2_m3")

nlmixr2_m3 %>%
  backfill_derived() %>%
  diagnose_constants(vol_pattern = "^V$")

nlmixr2_m3 %>%
  backfill_derived() %>%
  diagnose_constants(
    vol_pattern = "^V$",
    df_units = list(KA = "1/hr", ALPHA = "1/hr"),
    checks = list(neg_microvol = FALSE)
  )

# Using df form
derive_prm(nlmixr2_m3) %>%
  diagnose_constants(df = ., vol_pattern = "^V$")

## End(Not run)
```

diagram_lineage *Visualize* xpose_set

Description

[Experimental]

In its current state, this function is intended to provide a simple visual representation of an xpose_set. Functionality and aesthetic enhancements are expected in future releases.

Usage

```
diagram_lineage(xpdb_s, ...)
```

Arguments

xpdb_s	<xpose_set> object
...	For later expansion. Will be ignored.

Value

A DiagrammeR-compliant graph object.

Examples

```
## Not run:
diagram_lineage(pheno_set) %>%
  DiagrammeR::render_graph(layout="tree")

## End(Not run)
```

dv_vs_ipred_modavg *Model average plots*

Description

[Experimental]

This is for use when the model averaging of a set is planned.

Usage

```
dv_vs_ipred_modavg(
  xpdb_s,
  ...,
  .lineage = FALSE,
  algorithm = c("maa", "msa"),
  weight_type = c("individual", "population"),
  auto_backfill = FALSE,
  weight_basis = c("ofv", "aic", "res"),
  res_col = "RES",
  quiet
)

dv_vs_pred_modavg(
  xpdb_s,
  ...,
  .lineage = FALSE,
  algorithm = c("maa", "msa"),
  weight_type = c("individual", "population"),
  auto_backfill = FALSE,
  weight_basis = c("ofv", "aic", "res"),
  res_col = "RES",
  quiet
)

ipred_vs_idv_modavg(
  xpdb_s,
  ...,
  .lineage = FALSE,
  algorithm = c("maa", "msa"),
  weight_type = c("individual", "population"),
  auto_backfill = FALSE,
  weight_basis = c("ofv", "aic", "res"),
  res_col = "RES",
  quiet
)

pred_vs_idv_modavg(
  xpdb_s,
  ...,
  .lineage = FALSE,
  algorithm = c("maa", "msa"),
  weight_type = c("individual", "population"),
  auto_backfill = FALSE,
  weight_basis = c("ofv", "aic", "res"),
  res_col = "RES",
  quiet
)
```

```
plotfun_modavg(
  xpdb_s,
  ...,
  .lineage = FALSE,
  avg_cols = NULL,
  avg_by_type = NULL,
  algorithm = c("maa", "msa"),
  weight_type = c("individual", "population"),
  auto_backfill = FALSE,
  weight_basis = c("ofv", "aic", "res"),
  res_col = "RES",
  .fun = NULL,
  .funargs = list(),
  quiet
)
```

Arguments

xpdb_s	<xpose_set> object
...	<tidyselect> of models in set. If empty, all models are used in order of their position in the set. May also use a formula, which will just be processed with all.vars().
.lineage	<logical> where if TRUE, ... is processed
algorithm	<character> Model selection or model averaging
weight_type	<character> Individual-level averaging or by full dataset.
auto_backfill	<logical> If true, <backfill_iofv> is automatically applied.
weight_basis	<character> Weigh by OFV (default), AIC or residual.
res_col	<character> Column to weight by if "res" weight basis.
quiet	<logical> Minimize extra output.
avg_cols	<tidyselect> columns in data to average
avg_by_type	<character> Mainly for use in wrapper functions. Column type to average, but resulting column names must be valid for avg_cols (ie, same across all objects in the set). avg_cols will be overwritten.
.fun	<function> For slightly more convenient piping of model-averaged xpose_data into a plotting function.
.funargs	<list> Extra args to pass to function. If passing tidyselect arguments, be mindful of where quosures might be needed. See Examples.

Value

The desired plot

See Also

modavg_xpdb()

Examples

```
pheno_set %>%
  dv_vs_ipred_modavg(run8,run9,run10, auto_backfill = TRUE)

pheno_set %>%
  dv_vs_pred_modavg(run8,run9,run10, auto_backfill = TRUE)

pheno_set %>%
  ipred_vs_idv_modavg(run8,run9,run10, auto_backfill = TRUE)

pheno_set %>%
  pred_vs_idv_modavg(run8,run9,run10, auto_backfill = TRUE)

# Model averaged ETA covariates
pheno_set %>%
  plotfun_modavg(run8,run9,run10, auto_backfill = TRUE,
     avg_by_type = "eta",.fun = eta_vs_catcov,
     # Note quoting
     .funargs = list(etavar=quote(ETA1)))
```

---

eta_grid *Grid plots*

Description

This is essentially a wrapper around ggpairs, except it uses xpose motifs and styling. Note that this function produces a lot of repetitive output if quiet=FALSE; this may not be an issue, but it could look like an error has occurred if many covariates and individual parameter estimates are included.

Usage

```
eta_grid(
  xpdb,
  mapping = NULL,
  etavar = NULL,
  drop_fixed = TRUE,
  title = "Eta correlations | @run",
  subtitle = "Based on @nind individuals, Eta shrink: @etashk",
  caption = "@dir",
  tag = NULL,
  pairs_opts,
  .problem,
  quiet,
  ...
)
```

```
cov_grid(
  xpdb,
  mapping = NULL,
  cols = NULL,
  covtypes = c("cont", "cat"),
  show_n = TRUE,
  drop_fixed = TRUE,
  title = "Covariate relationships | @run",
  subtitle = "Based on @nind individuals",
  caption = "@dir",
  tag = NULL,
  pairs_opts,
  .problem,
  quiet,
  ...
)

eta_vs_cov_grid(
  xpdb,
  mapping = NULL,
  etavar = NULL,
  cols = NULL,
  covvar = NULL,
  covtypes = c("cont", "cat"),
  show_n = TRUE,
  drop_fixed = TRUE,
  title = "Eta covariate correlations | @run",
  subtitle = "Based on @nind individuals, Eta shrink: @etashk",
  caption = "@dir",
  tag = NULL,
  etacov = TRUE,
  pairs_opts,
  .problem,
  quiet,
  ...
)
```

Arguments

xpdb	<xp_xtras> or <xpose_data‘> object
mapping	ggplot2 style mapping
etavar	tidyselect for eta variables
drop_fixed	As in xpose
title	Plot title
subtitle	Plot subtitle
caption	Plot caption

tag	Plot tag
pairs_opts	List of arguments to pass to _opts. See <xplot_pairs>
.problem	Problem number
quiet	Silence extra debugging output
...	Passed to xplot_pairs
cols	tidyselect for covariates variables
covtypes	Subset to specific covariate type?
show_n	Count the number of IDs in each category
covvar	For eta_vs_cov_grid only: an alias for cols (matching the covvar argument of eta_vs_contcov()/eta_vs_catcov()). If supplied (non-NULL), takes precedence over cols.
etacov	Foreta_vs_cov_grid, eta are sorted after covariates to give an x orientation to covariate relationships.

Value

xp_tras_plot object

Examples

```
eta_grid(xpdb_x)
cov_grid(xpdb_x)
eta_vs_cov_grid(xpdb_x)

# Labels and units are also supported
xpdb_x %>%
  xpose::set_var_labels(AGE="Age", MED1 = "Digoxin") %>%
  xpose::set_var_units(AGE="yrs") %>%
  set_var_levels(SEX=lvl_sex(), MED1 = lvl_bin()) %>%
  eta_vs_cov_grid()
```

---

eta_vs_catcov *Eta categorical covariate plots (typical)*

Description

Eta categorical covariate plots (typical)

Usage

```
eta_vs_catcov(
  xpdb,
  mapping = NULL,
  etavar = NULL,
  covvar = NULL,
  drop_fixed = TRUE,
  orientation = "x",
  show_n = check_xpdb_x(xpdb, .warn = FALSE),
  type = "bol",
  list = TRUE,
  title = "Eta versus categorical covariates | @run",
  subtitle = "Based on @nind individuals, Eta shrink: @etashk",
  caption = "@dir",
  tag = NULL,
  facets,
  .problem,
  quiet,
  ...
)
```

Arguments

xpdb	<xp_xtras> or <xpose_data‘> object
mapping	ggplot2 style mapping
etavar	tidyselect for eta variables
covvar	tidyselect for categorical covariate variables; NULL (default) selects every categorical covariate in the xpdb data index.
drop_fixed	As in xpose
orientation	Passed to xplot_boxplot
show_n	Add "N=" to plot
type	Passed to xplot_boxplot
list	<logical> Only relevant when etavar resolves to more than one eta. If TRUE (default, for backwards compatibility), returns a plain list of one plot per eta. If FALSE, all etas are instead combined onto one shared plot – faceted by eta, in addition to the existing per-covariate facet – automatically paginating (at most 9 panels per page, i.e. ncol/nrow of 3) via xpose's own facet_wrap_paginate mechanism. Printing the returned plot renders every page; pass page to print() to select a specific one.
title	Plot title
subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag
facets	Additional facets

.problem	Problem number
quiet	Silence output
...	Any additional aesthetics.

Details

The ability to show number per covariate level is inspired by the package pmplots, but is implements here within the xpose ecosystem for consistency.

Value

The desired plot, or (when etavar resolves to more than one eta and list = TRUE) a plain list of one plot per eta.

Examples

```
eta_vs_catcov(xpdb_x)

# Labels and units are also supported
xpdb_x %>%
  xpose::set_var_labels(AGE="Age", MED1 = "Digoxin") %>%
  xpose::set_var_units(AGE="yrs") %>%
  set_var_levels(SEX=lvl_sex(), MED1 = lvl_bin()) %>%
  eta_vs_catcov()

# Combine all etas onto one shared, faceted plot instead of a list
eta_vs_catcov(xpdb_x, list = FALSE)

# Restrict to specific covariates with covvar, just like etavar
eta_vs_catcov(xpdb_x, covvar = SEX)
```

---

eta_vs_contcov *Eta continuous covariate plots (typical)*

Description

Eta continuous covariate plots (typical)

Usage

```
eta_vs_contcov(
  xpdb,
  mapping = NULL,
  etavar = NULL,
  covvar = NULL,
  drop_fixed = TRUE,
```

```
  linsm = FALSE,
  type = "ps",
  list = TRUE,
  title = "Eta versus continuous covariates | @run",
  subtitle = "Based on @nind individuals, Eta shrink: @etashk",
  caption = "@dir",
  tag = NULL,
  log = NULL,
  guide = TRUE,
  facets,
  .problem,
  quiet,
  ...
)
```

Arguments

xpdb	<xp_xtras> or <xpose_data‘> object
mapping	ggplot2 style mapping
etavar	tidyselect for eta variables
covvar	tidyselect for continuous covariate variables; NULL (default) selects every continuous covariate in the xpdb data index.
drop_fixed	As in xpose
linsm	If type contains "s" should the smooth method by lm?
type	Passed to xplot_scatter
list	<logical> Only relevant when etavar resolves to more than one eta. If TRUE (default, for backwards compatibility), returns a plain list of one plot per eta. If FALSE, all etas are instead combined onto one shared plot – faceted by eta, in addition to the existing per-covariate facet – automatically paginating (at most 9 panels per page, i.e. ncol/nrow of 3) via xpose's own facet_wrap_paginate mechanism. Printing the returned plot renders every page; pass page to print() to select a specific one.
title	Plot title
subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag
log	Log scale covariate value?
guide	Add guide line?
facets	Additional facets
.problem	Problem number
quiet	Silence output
...	Any additional aesthetics.

Value

The desired plot, or (when `etavar` resolves to more than one eta and `list = TRUE`) a plain list of one plot per eta.

Examples

```
eta_vs_contcov(xpdb_x)

# Labels and units are also supported
xpdb_x %>%
  xpose::set_var_labels(AGE="Age", MED1 = "Digoxin") %>%
  xpose::set_var_units(AGE="yrs") %>%
  set_var_levels(SEX=lvl_sex(), MED1 = lvl_bin()) %>%
  eta_vs_contcov()

# Combine all etas onto one shared, faceted plot instead of a list
eta_vs_contcov(xpdb_x, list = FALSE)

# Restrict to specific covariates with covvar, just like etavar
eta_vs_contcov(xpdb_x, covvar = AGE)
```

`expose_param` *Expose a model parameter of xpdb objects in an xpose_set*

Description

Expose a model parameter of xpdb objects in an xpose_set

Usage

```
expose_param(xpdb_s, ..., .problem = NULL, .subprob = NULL, .method = NULL)
```

Arguments

`xpdb_s`	<xpose_set> An xpose_set object
`...`	<dynamic-dots> One or more parameter to expose, using selection rules from `add_prm_association`.
`.problem`	<numeric> Problem number to apply this relationship.
`.subprob`	<numeric> Problem number to apply this relationship.
`.method`	<numeric> Problem number to apply this relationship.

Details

The parameter returned will be top-level, and to avoid conflicting names will be prepended by `..` (e.g., `..ome1`). The selector used to fetch the parameter will be used in this `..` name. If a better name is preferred, there are convenient renaming functions from `dplyr` where needed.

When using parameter selectors, quotations should be used for more complex names, like `"OMEGA(1,1)"`, since these may be read incorrectly otherwise.

The untransformed parameter is used for this exposure. The `get_prm` call uses `transform=FALSE`.

Value

An `xpose_set` object with the parameter exposed

See Also

`expose_property()`

Examples

```
pheno_set %>%
  expose_param(the1) %>%
  reshape_set()


pheno_set %>%
  expose_param(RUVADD, "OMEGA(1,1)") %>%
  reshape_set()

# This function is useful for generating a model-building table
pheno_set %>%
  # Determine longest lineage
  select(all_of(xset_lineage(.))) %>%
  # Select key variability parameters
  expose_param(RUVADD, "OMEGA(1,1)") %>%
  # Make sure all models have descriptions
  focus_qapply(desc_from_comments) %>%
  # Extract description
  expose_property(descr) %>%
  # Transform to tibble
  reshape_set() # %>% pipe into other processing
```

---

expose_property *Expose a property of xpdb objects in an xpose_set*

Description

Expose a property of xpdb objects in an xpose_set

Usage

```
expose_property(xpdb_s, ..., .problem = NULL, .subprob = NULL, .method = NULL)
```

Arguments

xpdb_s	<xpose_set> An xpose_set object
...	<dynamic-dots> One or more properties to expose
.problem	<numeric> Problem number to apply this relationship.
.subprob	<numeric> Problem number to apply this relationship.
.method	<numeric> Problem number to apply this relationship.

Details

The property returned will be top-level, and to avoid conflicting names will be prepended by .. (e.g., ..descr).

For some properties, transformations are applied automatically to make them more useful. This includes:

- etashk and epsshk: transformed to numeric vectors as in <get_shk>
- ofv and other per-problem properties: transformed as needed and pulls from each xpdb default problem.

Value

An xpose_set object with the properties exposed

See Also

expose_param()

Examples

```
xpdb_set <- expose_property(xpdb_set, descr)
xpdb_set$mod1$..descr

xpdb_set <- expose_property(xpdb_set, etashk)
xpdb_set$mod1$..etashk
```

focus_xpdb *Focus on an xpdb object in an xpose_set*

Description

For piping, set is passed, but with S3 method transformations are applied to the focused xpdb object.

Usage

```
focus_xpdb(xpdb_s, ..., .add = FALSE)

unfocus_xpdb(xpdb_s)

focused_xpdbs(xpdb_s)

focus_function(xpdb_s, fn, ...)

focus_qapply(xpdb_s, fn, ..., .mods = everything())
```

Arguments

xpdb_s	<xpose_set> An xpose_set object
...	<dynamic-dots> One or more xpdb objects to focus on
.add	<logical> Should the focus be added to the existing focus? (default: FALSE)
fn	<function> to apply to focused xpose_data objects
.mods	<tidyselect> Model names in set to quick-apply a function. See Details.

Details

While these functions are used internally, it is recognized that they may have value in user scripting. It is hoped these are self-explanatory, but the examples should address common uses.

Note: `focus_qapply()` (re)focuses as specified in `.mods` and then un-focuses all elements of the set so should only be used in the case where a quick application suffices. Otherwise, focusing with a sequence of `focus_function` calls (or a monolithic single `focus_function` call with a custom function) should be preferred.

`focus_function()`/`focus_qapply()` support two kinds of `fn`:

- *Transform* functions, which take an `xpose_data`/`xp_xtras` object and return one (e.g. `set_var_types_x`). These are applied to each focused element in place, and the (still-focused) `xpose_set` is returned so calls can keep being piped.
- *Output-generating* functions, which take an `xpose_data`/`xp_xtras` object but return something else (e.g. a plot or table). These are applied to each focused element, and the raw output is returned instead of an `xpose_set`: a single value if only one element is focused, or a named list (by label) of outputs if several are focused.

Value

An xpose_set object with the focused xpdb object(s) transformed in place, or, for functions that do not return an xpose_data/xp_xtras object, the output of fn (or a named list of outputs, if multiple elements are focused)

Examples

```
# Select two xpdb objects to focus on
xpdb_set %>% focus_xpdb(mod2,fix1)

# Add a focus
xpdb_set %>% focus_xpdb(mod2,fix1) %>% focus_xpdb(mod1, .add=TRUE)

# Remove focus
xpdb_set %>% focus_xpdb(mod2,fix1) %>% focus_xpdb()

## Not run:
# Focus function and tidyselect
pheno_set %>%
  focus_xpdb(everything()) %>%
  # Add iOFV col and iofv type to all xpdbs in set
  focus_function(backfill_iofv) %>%
  # Show 1... can do all like this, too, but no need
  unfocus_xpdb() %>%
  select(run6) %>%
  {.[[1]]$xpdb} %>%
  list_vars()

# Quick-apply version of previous example
pheno_set %>%
  focus_qapply(backfill_iofv) %>%
  select(run6) %>%
  {.[[1]]$xpdb} %>%
  list_vars()

## End(Not run)

# Output-generating function applied to a single focused element:
# returns the plot itself, not an xpose_set
pheno_set %>%
  focus_xpdb(run6) %>%
  focus_function(xpose::dv_vs_ipred)

# ... or with several elements focused, a named list of plots
pheno_set %>%
  focus_xpdb(run6, run7) %>%
  focus_function(xpose::dv_vs_ipred)
```

get_cov_matrix *Extract a parameter covariance or correlation matrix*

Description

Pulls the uncertainty (covariance step) matrix for estimated parameters, with built-in support for nonmem (via the .cov/.cor output tables) and nlmixr2 (via the fit object's covariance matrix) models. This is what feeds cormat(), but is exported separately since it may be useful on its own (eg, for programmatic checks on parameter colinearity).

Usage

```
get_cov_matrix(
  xpdb,
  type = c("correlation", "covariance"),
  .problem = NULL,
  .subprob = NULL,
  .method = NULL,
  drop_fixed = TRUE,
  quiet
)
```

Arguments

xpdb	<xp_xtras> or <xpose_data> object
type	<character> Either "correlation" (default) or "covariance"
.problem	<numeric> Problem number to use. Uses the xpose default if not provided. Ignored for nlmixr2 models.
.subprob	<numeric> Subproblem number to use. Uses the xpose default if not provided. Ignored for nlmixr2 models.
.method	<character> Method to use. Uses the xpose default if not provided. Ignored for nlmixr2 models.
drop_fixed	<logical> Drop fixed (or otherwise not estimated) parameters from the matrix. See Details.
quiet	<logical> Silence extra debugging output

Details

For nonmem models, the matrix is built from the .cor/.cov output tables produced by the $COV step. NONMEM includes fixed-effect parameters in these tables as placeholder zeros (since no uncertainty is estimated for them); drop_fixed (the default) removes them.

For nlmixr2 models, uncertainty is always calculated for fixed-effect (theta) parameters; whether it's also calculated for random-effect (omega) diagonal (variance) elements depends on the nlmixr2est version and fit settings – when present, those rows/columns are named "om.<eta name>" to distinguish them from the point estimates returned by get_prm(). drop_fixed has no effect here, since

parameters without a standard error are already excluded from the fit's covariance matrix rather than represented as placeholder zeros.

In both cases, if the covariance step was not run, or did not complete successfully, an informative error is raised rather than returning partial or placeholder data.

Value

A symmetric numeric matrix, with parameter names as `dimnames`.

Examples

```
get_cov_matrix(xpdb_x)
get_cov_matrix(xpdb_x, type = "covariance")

## Not run:
xpdb_nlmixr2 <- nlmixr_example("xpdb_nlmixr2")
get_cov_matrix(xpdb_nlmixr2)

## End(Not run)
```

`get_index` *Get full index for xpose_data data*

Description

Get full index for xpose_data data

Usage

```
get_index(xpdb, .problem = NULL, ...)

set_index(xpdb, index, ...)
```

Arguments

`xpdb`	<xpose_dataxpose::xpose_data> object
`.problem`	<numeric> Problem number to use. Uses the all problems if `NULL`
`...`	Ignored. Here for future expansion
`index`	<tibble> Index to set

Value

Tibble of index

Examples

```
get_index(xpose::xpdb_ex_pk)
```

get_prm *Access model parameters*

Description

Access model parameter estimates from an xpdb object.

Methods have been added to implement extensions. See Details.

Usage

```
get_prm(
  xpdb,
  .problem = NULL,
  .subprob = NULL,
  .method = NULL,
  digits = 4,
  transform = TRUE,
  show_all = FALSE,
  quiet
)
```

Arguments

xpdb	An xpose_data object from which the model output file data will be extracted.
.problem	The problem to be used, by default returns the last one for each file.
.subprob	The subproblem to be used, by default returns the last one for each file.
.method	The estimation method to be used, by default returns the last one for each file
digits	The number of significant digits to be displayed.
transform	Should diagonal OMEGA and SIGMA elements be transformed to standard deviation and off diagonal elements be transformed to correlations.
show_all	Logical, whether the 0 fixed off-diagonal elements should be removed from the output.
quiet	Logical, if FALSE messages are printed to the console.

Details

When using an <xp_xtra> object, this function will add a column to the output where CV% for each diagonal element of omega is calculated. This CV% is with respect to the resulting structural parameter, so unless the default log-normal association is applicable update with add_prm_association.

For log-normal, users may prefer to use the first-order CV% ($\sqrt{\omega^2}$) instead of the exact. In such case, xpdb <- set_option(xpdb, cvtype="sqrt") will get that preferred form.

If a single omega parameter is associated with multiple fixed effect parameters, the cv column will be a list. For the omega row associated with multiple fixed effect parameters, there will be

multiple CV values. This will be the case even if the transformation is log-normal and therefore scale-invariant, given the need for generality.

Note the approach used to calculate CV% assumes an untransformed scale for the fitted parameter value (unrelated to `transform`=TRUE). That means, for example, that for a logit-normal fitted parameter value, it is expected the value will be something constrained between 0 and 1, not the unbounded, continuous transformed value. The function <`mutate_prm`> is intended to help where that might be an issue.

Value

A tibble for single problem/subprob or a named list for multiple problem|subprob.

References

Prybylski, J.P. Reporting Coefficient of Variation for Logit, Box-Cox and Other Non-log-normal Parameters. Clin Pharmacokinet 63, 133-135 (2024). doi:10.1007/s40262023013432

See Also

`add_prm_association()`

Examples

```
# xpose parameter table
get_prm(xpose::xpdb_ex_pk, .problem = 1)

# xpose.xtra parameter table (basically the same)
get_prm(pheno_final, .problem = 1)

# For the sake of example, even though these were all lognormal:
pheno_final %>%
  add_prm_association(CLpkg~logit(IIVCL)) %>%
  add_prm_association(Vpkg~nmboxcox(IIVV, lambda = 0.01)) %>%
  get_prm(.problem = 1)
```

---

get_prm_nlmixr2 *get_prm equivalent for nlmixr2 fits*

Description

This is intended to match the <`xpose::get_prm`> rather than the updated `get_prm()`.

Usage

```
get_prm_nlmixr2(
  xpdb,
  transform = formals(get_prm)$transform,
  show_all = formals(get_prm)$show_all,
  quiet = FALSE
)
```

Arguments

xpdb	<xp_xtras> With nlmixr2 fit
transform	<logical> as in get_prm()
show_all	<logical> as in get_prm()
quiet	<logical> as in get_prm()

Value

a tibble with expected columns

---

get_prop *Generic function to extract a property from a model summary*

Description

Generic function to extract a property from a model summary

Usage

```
get_prop(
  xpdb,
  prop,
  .problem = NULL,
  .subprob = NULL,
  .method = NULL,
  .tail = 1
)
```

Arguments

xpdb	<xpose_dataxpose::xpose_data> object
prop	<character> Property to extract
.problem	<numeric> Problem number to use. Uses the xpose default if not provided.
.subprob	<numeric> Subproblem number to use. Uses the xpose default if not provided.
.method	<character> Method to use. Uses the xpose default if not provided.
.tail	<numeric> Length of terminal values to pull when there are more than 1 result

Value

Exact value for the property

Examples

```
data("xpdb_ex_pk", package = "xpose")

get_prop(xpdb_ex_pk, "descr")
```

`get_shk` *Get shrinkage estimates from model summary*

Description

This function parses shrinkages as they are currently presented in `get_summary`, so it is dependent on the current implementation of that function.

Usage

```
get_shk(xpdb, wh = "eta", .problem = NULL, .subprob = NULL, .method = NULL)
```

Arguments

`xpdb`	An `xpose_data` object.
`wh`	The shrinkage to extract (`"eta"` or `"eps"`)
`.problem`	Problem number to use. Uses the xpose default if not provided.
`.subprob`	`<numeric>` Subproblem number to use. Uses the xpose default if not provided.
`.method`	`<character>` Method to use. Uses the xpose default if not provided.

Value

A numeric vector of shrinkage estimates.

Examples

```
data("xpdb_ex_pk", package = "xpose")

# eta Shrinkage
get_shk(xpdb_ex_pk)

# epsilon Shrinkage
get_shk(xpdb_ex_pk, wh = "eps")
```

get_xtras_option *Inspect which* xpose.xtras *option value is dominant*

Description

For the two "two-tier" options – default_labs and default_watermark, which can be set both as a session-wide R option (via set_xtras_options()) and per-xpdb (via set_default_labs() / set_default_watermark()) – reports the value at each tier and which one is dominant, i.e. would currently be used by apply_default_labs() / add_watermark() absent any argument passed directly to those functions (which always takes precedence over both tiers).

The other options recognized by set_xtras_options() (save_dir, save_width, save_height, gg_theme, xp_theme) have no xpdb-level tier to compare against (see the Details in set_xtras_options()), so for those xpdb is always NULL and dominant is "option" or "neither".

Usage

```
get_xtras_option(name, xpdb = NULL)
```

Arguments

name	<character> One of the option names recognized by set_xtras_options()
xpdb	<xpose_data> or <xp_xtras> object to check for an xpdb-level value (optional; only consulted for default_labs/default_watermark)

Value

a list with elements option (the getOption() value), xpdb (the xpdb-level value, or NULL), and dominant (one of "option", "xpdb", or "neither")

Examples

```
options(xpose.xtras.default_labs = list(caption = "session default"))
xpdb_x2 <- set_default_labs(xpdb_x, caption = "model-specific")
get_xtras_option("default_labs", xpdb_x2)
options(xpose.xtras.default_labs = NULL)
```

ggsave_xp *Save a plot with* xpose.xtras *default output resolution*

Description

A ggplot2::ggsave()-compatible wrapper (defaulting to ggplot2::ggsave() itself, but swappable via save_fun – e.g. for reportifyr::ggsave_with_metadata() or any other function sharing ggsave()'s plot/filename/path/width/height signature). Before saving: resolved labels are applied via apply_default_labs() and a configured watermark (if any) via add_watermark() – see apply_labs/apply_watermark, both of which default to the xpose.xtras.auto_apply option (TRUE unless changed), the same switch print.xpose_plot() uses, so a plot looks the same whether it's viewed interactively or saved with ggsave_xp(). filename/path have any @keyword placeholders expanded via xpose::parse_title(), the same way xpose::xpose_save() does (independent of which save_fun is used, since most save functions don't do this themselves); and path/width/height/save_fun fall back to the xpose.xtras.save_dir, xpose.xtras.save_width, xpose.xtras.save_height, and xpose.xtras.save_fun R options when not supplied explicitly, so a project can set output defaults once (see set_xtras_options()).

Usage

```
ggsave_xp(
  plot = ggplot2::last_plot(),
  filename,
  path = getOption("xpose.xtras.save_dir"),
  width = getOption("xpose.xtras.save_width", 7),
  height = getOption("xpose.xtras.save_height", 6),
  xpdb = NULL,
  apply_labs = getOption("xpose.xtras.auto_apply", TRUE),
  apply_watermark = getOption("xpose.xtras.auto_apply", TRUE),
  save_fun = getOption("xpose.xtras.save_fun", ggplot2::ggsave),
  ...
)
```

Arguments

plot	<ggplot> or <xpose_plot> object
filename	<character> File name, optionally with @keyword placeholders (e.g. "@run_@plotfun.pdf", see xpose::parse_title()). @plotfun (the name of the function that built plot, e.g. "dv_vs_ipred") only resolves when plot is an xpose_plot, since it isn't something a bare xpdb knows about
path	<character> Directory to save in; falls back to the xpose.xtras.save_dir R option
width, height	<numeric> Plot size (in inches by default, see save_fun's own units argument if it has one); fall back to the xpose.xtras.save_width/xpose.xtras.save_height R options
xpdb	<xpose_data> or <xp_xtras> object plot was built from, forwarded to apply_default_labs()/add_watermark() (see their xpdb argument) and used to resolve @keyword placeholders
apply_labs	<logical> Apply apply_default_labs() to plot before saving; defaults to the xpose.xtras.auto_apply option (TRUE unless changed)

apply_watermark
: <logical> Apply add_watermark() to plot before saving, if a default_watermark is configured at some tier (see add_watermark()) – otherwise a no-op regardless; defaults to the xpose.xtras.auto_apply option (TRUE unless changed)

save_fun
: <function> The actual save function to call, e.g. ggplot2::ggsave() (the default) or a drop-in alternative such as reportifyr::ggsave_with_metadata(); falls back to the xpose.xtras.save_fun R option, then ggplot2::ggsave()

...
: Passed on to save_fun (e.g. device, dpi, units, bg)

Value

the result of save_fun (for the default ggplot2::ggsave(), the saved file path, invisibly)

See Also

set_xtras_options() for the full list of xpose.xtras.* session options.

Examples

```
## Not run:
options(xpose.xtras.save_dir = "figures", xpose.xtras.save_width = 8)
p <- xpose::dv_vs_ipred(xpose::xpdb_ex_pk)
ggsave_xp(p, filename = "dv_vs_ipred.png")

## End(Not run)
```

---

grab_xpose_plot *Grab processed* xpose_plot

Description

This function is very simple and unlikely to capture every possible situation. Paginated plots are not supported.

This is helpful for working with xpose plots in patchwork or ggpubr functions.

Usage

```
grab_xpose_plot(plot)
```

Arguments

plot
: <xpose_plot> or list thereof

Value

Grob or list of grobs

Examples

```
single_plot <- xpdb_x %>%
eta_vs_catcov(etavar = ETA1) %>%
grab_xpose_plot()

listof_plots <- xpdb_x %>%
eta_vs_catcov(etavar = c(ETA1,ETA3)) %>%
grab_xpose_plot()
```

`ind_plots_sample` *Individual plots for a (stratified) sample of individuals*

Description

A wrapper around `ind_plots` that first draws a sample of `n` individuals (9 by default, enough to fill a 3x3 page) rather than plotting every individual in the dataset. If `stratify` is provided, the sample is drawn proportionally from each level (or combination of levels) of the `tidyselect`-ed column(s), so the sample remains as representative as the data and `n` allow.

Usage

```
ind_plots_sample(
  xpdb,
  n = 9,
  stratify = NULL,
  seed = NULL,
  facets,
  .problem,
  quiet,
  ...
)
```

Arguments

`xpdb`	<xp_xtras> or <xpose_data> object
`n`	<integer> Number of individuals to sample. Defaults to 9. If fewer individuals than `n` are available, all of them are used.
`stratify`	<tidyselect> Optional column(s), other than the id column, to stratify the sample by.
`seed`	<integer> Optional seed, set (and restored on exit) for reproducible sampling.
`facets`	As in `ind_plots`. Defaults to the id column (and `stratify` column(s), if given) added to `xpdb$xp_theme$facets`.
`.problem`	<numeric> Problem number to use.
`quiet`	<logical> Silence extra output.
`...`	Passed on to `ind_plots`

Details

When `stratify` is used, the stratifying column(s) are appended to the facet formula (in addition to the id column that `ind_plots` already facets by), so that the stratum each sampled individual belongs to is visible in the plot.

Stratified sample sizes are allocated proportionally to stratum size using the largest-remainder method, so the total sampled always equals `min(n, sum(individuals available across all strata))`.

Value

The desired plot

See Also

`ind_roc()`

Examples

```
xpdb_x %>% ind_plots_sample(n = 6)
xpdb_x %>% ind_plots_sample(n = 6, stratify = SEX)
```

---

ind_roc *Individual ROC plots*

Description

To identify any individual likelihood predictions that may be more influential or unusual.

Note this function may have a long runtime.

Usage

```
ind_roc(
  xpdb,
  mapping = NULL,
  cutpoint = 1,
  type = "ca",
  title = "Individual ROC curves | @run",
  subtitle = "Ofv: @ofv, Eps shrink: @epsshk",
  caption = "@dir | Page @page of @lastpage",
  tag = NULL,
  facets,
  .problem,
  quiet,
  ...
)
```

Arguments

xpdb	<xp_xtras> or <xpose_data> object
mapping	ggplot2 style mapping
cutpoint	<numeric> Of defined probabilities, which one to use in plots.
type	See Details.
title	Plot title
subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag
facets	Additional facets
.problem	Problem number
quiet	Silence extra debugging output
...	Any additional aesthetics.

Details

For type-based customization of plots:

- c ROC curve (using geom_path)
- k Key points on ROC curve (where on curve the threshold is thres_fixed) (using geom_point)
- p ROC space points (using geom_point)
- t ROC space text (using geom_text)
- a AUC in bottom right (using geom_label)

Value

The desired plot

Examples

```
## Not run:
vismo_pomod  %>%
  set_var_types(.problem=1, catdv=DV, dvprobs=matches("^P\\d+$")) %>%
  set_dv_probs(.problem=1, 0~P0,1~P1,ge(2)~P23) %>%
  ind_roc()

## End(Not run)
```

iofv_vs_mod *Objective function changes across models*

Description

Another visualization of how individual objective functions change over the course of model development.

Usage

```
iofv_vs_mod(
  xpdb_s,
  ...,
  .lineage = FALSE,
  auto_backfill = FALSE,
  mapping = NULL,
  orientation = "x",
  type = "bjc",
  title = "Individual Ofvs across models",
  subtitle = "Based on @nind individuals, Initial Ofv: @ofv",
  caption = "Initial @dir",
  tag = NULL,
  axis.text = "@run",
  facets,
  .problem,
  quiet
)
```

Arguments

xpdb_s	<xpose_set> object
...	<tidyselect> of models in set. If empty, all models are used in order of their position in the set. May also use a formula, which will just be processed with all.vars().
.lineage	<logical> where if TRUE, ... is processed
auto_backfill	<logical> If TRUE, apply <backfill_iofv()> automatically. FALSE by default to encourage data control as a separate process to plotting control.
mapping	ggplot2 style mapping
orientation	Defaults to x
type	Passed to <xplot_boxplot>
title	Plot title
subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag

axis.text	What to label the model. This is parsed on a per-model basis.
facets	Additional facets
.problem	Problem number
quiet	Silence output

Value

The desired plot

Examples

```
pheno_set %>%
  focus_qapply(backfill_iofv) %>%
  iofv_vs_mod()

pheno_set %>%
  focus_qapply(backfill_iofv) %>%
  iofv_vs_mod(run3,run11,run14,run15)

pheno_set %>%
  focus_qapply(backfill_iofv) %>%
  iofv_vs_mod(.lineage = TRUE)
```

---

ipred_vs_ipred *Compare model predictions*

Description

For two models in an xpose_set, these functions are useful in comparing individual and population predictions

Usage

```
ipred_vs_ipred(
  xpdb_s,
  ...,
  .inorder = FALSE,
  type = "pls",
  title = "Individual prediction comparison | @run",
  subtitle = "Ofv: @ofv, Eps shrink: @epsshk",
  caption = "@dir",
  tag = NULL,
  log = NULL,
  guide = TRUE,
  axis.text = "@run",
```

```
  facets,
  .problem,
  quiet
)

pred_vs_pred(
  xpdb_s,
  ...,
  .inorder = FALSE,
  type = "pls",
  title = "Population prediction comparison | @run",
  subtitle = "Ofv: @ofv, Eps shrink: @epsshk",
  caption = "@dir",
  tag = NULL,
  log = NULL,
  guide = TRUE,
  axis.text = "@run",
  facets,
  .problem,
  quiet
)
```

Arguments

xpdb_s	<xpose_set> object
...	See <two_set_dots>
.inorder	See <two_set_dots>
type	Passed to xplot_scatter
title	Plot title
subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag
log	Log scale covariate value?
guide	Add guide line?
axis.text	What to show in the axes to distinguish the model values
facets	Additional facets
.problem	Problem number
quiet	Silence output

Value

The desired plot

Examples

```
pheno_set %>%
  ipred_vs_ipred(run5,run15)

pheno_set %>%
  pred_vs_pred(run5,run15)
```

`irep` *Add simulation counter*

Description

Add a column containing a simulation counter (irep). A new simulation is counted every time a value in x is different than its previous value and is a duplicate.

xpose fixed this upstream around version 0.5.0, then later reverted that fix, so this is treated as a standing bugfix rather than a temporary one pending removal (previously this deferred to `xpose::irep()` for xpose >= 0.5.0; that's no longer safe to assume).

This version of the function does not require IDs be ascending, but does not work for datasets where IDs are repeated (not in sequence). Both cases are read as separate individuals for NONMEM, but NONMEM does not need to detect repetition of ID sequences (for NONMEM, `1,1,2,2,3,3,1,1,2,2,3,3` is 6 individuals, regardless of being 2 repeats of 3 individuals). Given the vast majority of datasets use 1 individual per ID, (which cannot be said about IDs always being ascending), only one of these corrections is implemented.

Usage

```
irep(x, quiet = FALSE)
```

Arguments

x	The column to be used for computing simulation number, usually the ID column.
quiet	Logical, if FALSE messages are printed to the console.

Details

Bugfix for irep.

Value

<numeric> vector tracking the number of simulations based on unique subject IDs.

Examples

```
data("xpdb_ex_pk", package = "xpose")

xpdb_ex_pk_2 <- xpdb_ex_pk %>%
 mutate(sim_id = irep(ID), .problem = 2)
```

is_xp_xtras *Basic class checker for* xp_xtras

Description

Basic class checker for xp_xtras

Usage

```
is_xp_xtras(x)
```

Arguments

x	Object to test

Value

<logical> TRUE if xp_xtras object

Examples

```
is_xp_xtras(xpose::xpdb_ex_pk)

is_xp_xtras(xpdb_x)
```

left_join_x *Backfill missing variables via a left join*

Description

[Experimental]

<dplyr::left_join> wrapper for xpose_data (and, by inheritance, xp_xtras) objects. Unlike a plain left_join(), a column present in both x and y (other than the join keys) is not duplicated with .x/.y suffixes: missing (NA) values already in x are backfilled from the matching value in y, while non-missing values already in x are left untouched. This makes it straightforward to backfill a variable (or set of variables) that is only partially recorded, from a second data source keyed on the same join variable(s) (e.g. ID).

left_join_x() accepts xpose_data/xp_xtras objects directly, with an additional .problem argument restricting which problem(s) the join is applied to.

left_join() without _x is defined as an S3 method on xpose_data, so that the usual <dplyr::left_join> generic dispatches here automatically (xp_xtras objects are handled the same way, via class inheritance).

Usage

```
left_join_x(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = NULL,
  .problem = NULL
)

## S3 method for class 'xpose_data'
left_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = NULL,
  .problem = NULL
)
```

Arguments

x	An xpose_data or xp_xtras object.
y	A data frame (or another object coercible to one) to join in.
by	Join specification, as in <dplyr::left_join>. If NULL, a natural join is performed using variables common to x and y.
copy	If x and y are not from the same source and copy = TRUE, y is copied to bring it into the same source as x. See <dplyr::left_join>.
suffix	Suffixes used internally to disambiguate a column shared by x and y before it is backfilled into a single column; not visible in the result.
...	Other parameters passed onto <dplyr::left_join>.
keep	Passed to <dplyr::left_join>. Note that duplicate join key columns (keep = TRUE) are backfilled together like any other shared column, rather than kept separate.
.problem	The problem number(s) to which the join will be applied. Uses all problems if NULL.

Value

An updated xpose_data/xp_xtras object.

Examples

```
# Some subjects are missing an APGR score in the base dataset
xpdb_missing <- pheno_base %>%
  mutate_x(APGR = dplyr::if_else(ID %in% c("1", "2"), NA, APGR))

# A separate table with the (complete) values, keyed on ID
apgr_lookup <- xpose::get_data(pheno_base, quiet = TRUE) %>%
  dplyr::distinct(ID, APGR)

# Existing APGR values are kept; only the missing ones are filled in
left_join_x(xpdb_missing, apgr_lookup, by = "ID")
```

---

list_dv_probs *For a categorical DV variable, show associated probabilities*

Description

A convenient quick check for how probabilities are currently assigned, based on set_dv_probs.

Usage

```
list_dv_probs(xpdb, .problem = NULL, .dv_var = NULL)
```

Arguments

xpdb	<xp_xtras> object
.problem	<numeric> Problem number to use. Uses all problems if NULL (the default). May be omitted entirely and left to default, even when formulas are supplied positionally in ....
.dv_var	<tidyselect> of column having the categorical observation. Default is first-listed catdv.

Value

<tibble> of probabilities

Examples

```
pkpd_m3 %>%
  set_dv_probs(1, 1~LIKE, .dv_var = BLQ) %>%
  list_dv_probs(.dv_var=BLQ)
```

list_vars *Updates to* list_vars

Description

To accommodate changes made in xpose.xtras, <list_vars> needed some minimal updates.

Usage

```
list_vars(xpdb, .problem = NULL, ...)

## Default S3 method:
list_vars(xpdb, .problem = NULL, ...)

## S3 method for class 'xp_xtras'
list_vars(xpdb, .problem = NULL, ...)
```

Arguments

xpdb	<xpose_data> or <xp_xtras> object
.problem	<numeric> Problem number to use. Uses the all problems if NULL
...	Should be blank.

Value

<tibble> of variables

Examples

```
list_vars(xpose::xpdb_ex_pk)

list_vars(xpdb_x)
```

logLik.xpose_data *Log-likelihood, AIC and BIC for* xpose_data *objects*

Description

[Experimental]

NONMEM (and nlmixr2) objective function values (OFV) are -2*log-likelihood, so the log-likelihood of a model is derived as -ofv/2. The number of estimated parameters (thetas, and unfixed omegas/sigmas) is used as the degrees of freedom.

Because <stats::AIC> and <stats::BIC> dispatch through <stats::logLik> by default, only logLik.xpose_data needs to be defined here for AIC()/BIC() to work as expected on xpose_data/xp_xtras objects; no AIC/BIC methods are defined for a single model.

Not calculable for a model-averaged ("franken") xpose_data object (eg, the output of <modavg_xpdb>), since such an object does not correspond to a single fitted model; an error is thrown instead.

Usage

```
## S3 method for class 'xpose_data'
logLik(object, .problem = NULL, .subprob = NULL, .method = NULL, ...)
```

Arguments

object	<xpose_data> or <xp_xtras> object
.problem	<numeric> Problem number to use. Uses the xpdb default if not provided.
.subprob	<numeric> Subproblem number to use. Uses the xpdb default if not provided.
.method	<numeric> Method to use. Uses the xpdb default if not provided.
...	Not used.

Value

<logLik> object, as documented in <stats::logLik>, with "df" (number of estimated parameters) and "nobs" (number of observations) attributes set.

Examples

```
logLik(xpdb_x)
AIC(xpdb_x)
BIC(xpdb_x)
```

---

logLik.xpose_set *Log-likelihood, AIC and BIC across an* xpose_set

---

Description

[Experimental]

If no base model is provided, and if lineage is unclear, the first model in the xpose_set is used as the base model, exactly as in <diff.xpose_set>. Unlike diff(), values are not differenced, so a straightforward model-to-model comparison is possible.

As with the xpose_data methods, a component model that is itself a model-averaged ("franken") object will cause an error.

Usage

```
## S3 method for class 'xpose_set'
logLik(object, ...)

## S3 method for class 'xpose_set'
AIC(object, ..., k = 2)

## S3 method for class 'xpose_set'
BIC(object, ...)
```

Arguments

object	<xpose_set> object
...	<dynamic-dots> Passed to <xset_lineage>. .spinner=FALSE can also be set here.
k	<numeric> Penalty per parameter, as in <stats::AIC>.

Value

<numeric> vector, or list thereof, following <xset_lineage>.

---

modavg_xpdb *Create a model-averaged xpose data object*

Description

[Experimental]

This function is a helper for plotting functions where models in an xpose_set can be averaged together. The implementation attempts to match and extend from the cited prior work.

Usage

```
modavg_xpdb(
  xpdb_s,
  ...,
  .lineage = FALSE,
  avg_cols = NULL,
  avg_by_type = NULL,
  algorithm = c("maa", "msa"),
  weight_type = c("individual", "population"),
  auto_backfill = FALSE,
  weight_basis = c("ofv", "aic", "res"),
  res_col = "RES",
  quiet
)
```

Arguments

xpdb_s	<xpose_set> object
...	<tidyselect> of models in set. If empty, all models are used in order of their position in the set. May also use a formula, which will just be processed with all.vars().
.lineage	<logical> where if TRUE, ... is processed
avg_cols	<tidyselect> columns in data to average
avg_by_type	<character> Mainly for use in wrapper functions. Column type to average, but resulting column names must be valid for avg_cols (ie, same across all objects in the set). avg_cols will be overwritten.
algorithm	<character> Model selection or model averaging
weight_type	<character> Individual-level averaging or by full dataset.
auto_backfill	<logical> If true, <backfill_iofv> is automatically applied.
weight_basis	<character> Weigh by OFV (default), AIC or residual.
res_col	<character> Column to weight by if "res" weight basis.
quiet	<logical> Minimize extra output.

Value

Weight-averaged <xpose_data> object.

References

Uster, D.W., Stocker, S.L., Carland, J.E., Brett, J., Marriott, D.J.E., Day, R.O. and Wicha, S.G. (2021), A Model Averaging/Selection Approach Improves the Predictive Performance of Model-Informed Precision Dosing: Vancomycin as a Case Study. Clin. Pharmacol. Ther., 109: 175-183. doi:10.1002/cpt.2065

Examples

```
pheno_set %>%
  modavg_xpdb(
    avg_cols = IPRED,
    auto_backfill = TRUE,
    algorithm = "maa",
    weight_basis = "aic"
  )
```

modify_xpdb *Add, remove or rename variables in an xpdb*

Description

`mutate_x()` adds new variables and preserves existing ones. `select()` keeps only the listed variables; `rename()` keeps all variables.

Note: this function uses `xpose.xtras::edit_xpose_data`, but is otherwise the same as <`xpose::mutate`>.

Usage

```
mutate_x(.data, ..., .problem, .source, .where)

rename_x(.data, ..., .problem, .source, .where)
```

Arguments

`.data`	An xpose database object.
`...`	Name-value pairs of expressions. Use `NULL` to drop a variable.
`.problem`	The problem from which the data will be modified
`.source`	The source of the data in the xpdb. Can either be 'data' or an output file extension e.g. 'phi'.
`.where`	A vector of element names to be edited in special (e.g. `.where = c('vpc_dat', 'aggr_obs')` with vpc).

Value

An updated `xpose` data object

mutate_prm *Transform parameter values in place*

Description

Apply transformations to fitted parameter values.

As fitted, sometimes parameter values are not as easy to communicate, but to transform them outside of the `xpose` ecosystem limits some available features. To have the best experience, this function can update the parameter values that are used by `xpose get_prm` functions. At this time these transformations are not applied to `param` vars (`list_vars`), but that can already be done with the `mutate` method.

This only works for theta parameters.

All valid mutations are applied sequentially, so a double call to `the2~the2^3` will result in effectively `the2~the2^9`, for example.

RSE values are calculated at runtime within `get_prm`, so they are not updated (or updatable) with this function.

Usage

```
mutate_prm(
  xpdb,
  ...,
  .autose = TRUE,
  .problem = NULL,
  .subprob = NULL,
  .method = NULL,
  .sesim = 1e+05,
  quiet
)
```

Arguments

xpdb	<xp_xtras> object
...	... <dynamic-dots> One or more formulae that define transformations to parameters. RHS of formulas can be function or a value. That value can be a function call like in mutate() (the1~exp(the1)).
.autose	<logical> If a function is used for the transform then simulation is used to transform the current SE to a new SE. Precision of this transformation is dependent on .sesim. If parameter values are not assigned with a function, this option will simply scale SE to maintain the same RSE. See Details.
.problem	<numeric> Problem number to apply this relationship.
.subprob	<numeric> Problem number to apply this relationship.
.method	<numeric> Problem number to apply this relationship.
.sesim	<numeric> Length of simulated rnorm vector for .autose.
quiet	Silence extra output.

Details

Important points about covariance and correlation (for NONMEM only):

Covariance and correlation parameters are adjusted when standard error (SE) values are changed directly or with .autose. When a transformation is applied as a function for the fixed effect parameter (eg, ~plogis), the resulting SE may have an unexpected scale; this is because it is now reporting the standard deviation of a transformed and potentially non-normal distribution. If the parameter were fit in the transformed scale (constrained to any appropriate bounds), it would likely have a different SE given that most covariance estimation methods (excluding non-parametric and resampling-based) will treat the constrained parameter as continuous and unconstrained.

The updates to variance-covariance values (and the correlation values, though that is mostly invariant) are applied to the entire matrices. When piped directly into get_prm, only the SE estimate is shown, but <get_file> can be used to see the complete updated variance-covariance values. This could be useful if those matrices are being used to define priors for a Bayesian model fitting, as the re-scaling of off-diagonal elements is handled automatically.

For all software: A function to transform parameters will result in a more accurate autose result. If a call (the1~exp(the)) or a value (the1~2) are used, the standard error will be simply scaled.

Value

An updated xp_xtras object with mutated parameters

Examples

```
vismo_pomod %>%
  # Function
  mutate_prm(THETA11~exp) %>%
  # Value (se will not be scaled); plogis = inverse logit
  mutate_prm(THETA12~plogis(THETA12)) %>%
  get_prm()
```

---

nlmixr2_as_xtra *Convenience function for ingesting an nlmixr2 model to xpose and xpose.xtras*

Description

A wrapper that executes the pipeline:

```
obj %>%
 xpose.nlmixr2::xpose_data_nlmixr2() %>%
 attach_nlmixr2() %>%
 as_xp_xtras() %>%
 backfill_nlmixr2_props() %>%
 `if`(.skip_assoc, ., nlmixr2_prm_associations(.))
```

Usage

```
nlmixr2_as_xtra(obj, ..., .skip_assoc = FALSE)
```

Arguments

obj	nlmixr2 fit object
...	Passed to xpose_data_nlmixr2
.skip_assoc	<logical> If the model is relatively uncomplicated, nlmixr2_prm_associations() may be able to recognize relationships between random effects and fixed effect parameters. If the default (FALSE) fails then try to rerun with the association step skipped.

Value

An <xp_xtra> object with fit attached

See Also

attach_nlmixr2()

`nlmixr2_prm_associations` *Based on associations baked into nlmixr2, automatically add to xpose data*

Description

This function attempts to discern the associations between omegas and thetas using information about mu referencing within the `nlmixr2` fit object.

Usage

```
nlmixr2_prm_associations(xpdb, dry_run = FALSE, quiet)
```

Arguments

`xpdb`	<xp_xtras> object
`dry_run`	<logical> Return a resulting information to compare against.
`quiet`	<logical> Include extra information

Details

Back-transformations are not as relevant here as they may seem. Manual back-transformation with `backTransform()` only affects the display of the back-transformed theta estimate (and CI), but does not impact the relationship between EBEs and individual parameter estimates.

Value

Object with filled `par`

See Also

`rxode2::ini()`

Examples

```
## Not run:
nlmixr2_warfarin <- nlmixr_example("nlmixr2_warfarin")

nlmixr2_warfarin %>%
  # This will add all log-normal and the logitnormal params
  nlmixr2_prm_associations() %>%
  # Make sure theta is in normal scale
  # rxode::expit could be plogis in this case
  mutate_prm(temax ~ rxode2::expit) %>%
  # Review results
  get_prm()
```

```
## End(Not run)
```

`nlmixr_example` *Generate example* `xp_xtras` *objects from nlmixr2 fits*

Description

Runs an nlmixr2 fit on demand and returns the result as a ready-to-use `xp_xtras` object. `nlmixr2_example` is an alias.

Usage

```
nlmixr_example(name)

nlmixr2_example(name)
```

Arguments

`name`	`<character>` Name of the example to generate. One of `"xpdb_nlmixr2"`, `"xpdb_nlmixr2_saem"`, `"nlmixr2_warfarin"`, `"nlmixr2_m3"`, `"xpdb_nlmixr2_nocov"`.

Details

Available examples:

`"xpdb_nlmixr2"` One-compartment FOCEI fit to the theophylline dataset. The basic introductory example from the nlmixr2 documentation. See Theoph.

`"xpdb_nlmixr2_saem"` The same one-compartment theophylline model fit with SAEM instead of FOCEI.

`"nlmixr2_warfarin"` Multiple-endpoint warfarin PK/PD model with a four-compartment PK and turnover PD. Provides categorical covariate data useful for eta diagnostic examples.

`"nlmixr2_m3"` Theophylline one-compartment model with censoring applied to provoke M3 likelihood handling. Includes a `BLQLIKE` output variable for use as a categorical DV example with `catdv_vs_dvprobs()`.

`"xpdb_nlmixr2_nocov"` The same one-compartment theophylline FOCEI fit as `"xpdb_nlmixr2"`, but with the covariance step skipped (`covMethod = ""`). Useful for exercising code paths that depend on parameter uncertainty, such as `get_cov_matrix()`, when it is not available.

Value

An `xp_xtras` object with the nlmixr2 fit attached.

Source

"xpdb_nlmixr2" and "xpdb_nlmixr2_saem": https://nlmixr2.org/articles/running_nlmixr.html

"nlmixr2_warfarin": https://nlmixr2.org/articles/multiple-endpoints.html

"nlmixr2_m3": https://github.com/nlmixr2/nlmixr2/issues/275#issuecomment-2445469327

References

Fidler M (2025). *nlmixr2: Nonlinear Mixed Effects Models in Population PK/PD*. doi:10.32614/CRAN.package.nlmixr2, R package version 3.0.2, https://CRAN.R-project.org/package=nlmixr2.

Fidler M, Wilkins J, Hooijmaijers R, Post T, Schoemaker R, Trame M, Xiong Y, Wang W (2019). "Nonlinear Mixed-Effects Model Development and Simulation Using nlmixr and Related R Open-Source Packages." *CPT: Pharmacometrics & Systems Pharmacology*, *8*(9), 621-633. doi:10.1002/psp4.12445.

Schoemaker R, Fidler M, Laveille C, Wilkins J, Hooijmaijers R, Post T, Trame M, Xiong Y, Wang W (2019). "Performance of the SAEM and FOCEI Algorithms in the Open-Source, Nonlinear Mixed Effect Modeling Tool nlmixr." *CPT: Pharmacometrics & Systems Pharmacology*, *8*(12), 923-930. doi:10.1002/psp4.12471.

Beal, S.L. Ways to Fit a PK Model with Some Data Below the Quantification Limit. J Pharmacokinet Pharmacodyn 28, 481-504 (2001). doi:10.1023/A:1012299115260

See Also

nlmixr2_as_xtra(), catdv_vs_dvprobs()

Examples

```
## Not run:
xpdb_nlmixr2 <- nlmixr_example("xpdb_nlmixr2")

nlmixr2_m3 <- nlmixr_example("nlmixr2_m3")
nlmixr2_m3 %>%
  set_var_types(catdv = CENS, dvprobs = BLQLIKE) %>%
  set_dv_probs(1, 1 ~ BLQLIKE, .dv_var = CENS) %>%
  set_var_levels(1, CENS = lvl_bin()) %>%
  catdv_vs_dvprobs(xlab = "basic", quiet = TRUE)

## End(Not run)
```

---

normalize_etas *Normalize etas by their omega- or empirical-SD-implied scale*

Description

Sets xpdb$normalize_etas, a top-level slot (alongside eg $covs, see add_cov_association()) consumed by eta_grid()/ eta_vs_cov_grid()/eta_vs_contcov()/eta_vs_catcov(): each selected eta is divided by its typical scale – by default the standard deviation implied by its associated diagonal omega estimate (sqrt(omega)), same as recalc_shk() uses – before being plotted, so etas modeled on very different scales (eg a normally-distributed eta next to a log-normal one with a much larger omega) can be compared on one shared plot without the larger-scale eta dominating.

normalise_etas() is an alias, for the British/rest-of-world spelling.

Usage

```
normalize_etas(
  xpdb,
  ...,
  .use_sd = FALSE,
  .problem = NULL,
  .subprob = NULL,
  .method = NULL,
  quiet
)

normalise_etas(
  xpdb,
  ...,
  .use_sd = FALSE,
  .problem = NULL,
  .subprob = NULL,
  .method = NULL,
  quiet
)
```

Arguments

xpdb	<xpose_dataxpose::xpose_data> or xp_xtras object
...	<tidyselect> Which eta column(s) to (re)compute a normalization factor for. Defaults to every eta column for .problem.
.use_sd	<logical> If TRUE, normalize by the empirical standard deviation of each eta's individual estimates instead of the omega-implied one; see Details. Defaults to FALSE.
.problem	<numeric> Problem number to use. Uses the xpose default if not provided.
.subprob	<numeric> Subproblem number to use. Uses the xpose default if not provided.
.method	<character> Method to use. Uses the xpose default if not provided.
quiet	<logical> Silence extra debugging output

Details

This only ever affects how those four plotting functions *display* etas – it never modifies `xpdb$data`, so `get_data()` and every other consumer of the eta columns keep seeing the raw (unnormalized) values.
`$normalize_etas` is a plain top-level slot rather than an `xpdb$options` entry – unlike most options, its value is one number per eta rather than a single setting, and folding a handful of high-precision numbers per eta into `print.xpose_data()`'s single-line `Options:` summary made that summary unreadable for models with more than a couple of etas.

The default (omega-based) scale relies on the same internal name/numbering match between eta columns and diagonal omega estimates (see `recalc_shk()`'s Details for when that can fail, eg unconventional eta naming that isn't a `nlmixr2`-style direct match to a parameter table `name` and also doesn't follow NONMEM's `ETA<k>`/`ETA(k)` numbering). When that match fails, or there simply is no reliable omega for these etas (eg a hand-built or simulated `xpdb`), `.use_sd = TRUE` sidesteps it entirely, scaling by the empirical standard deviation of each eta's own individual estimates instead – at the cost of that scale itself being sample-dependent (and shrinkage- deflated) rather than reflecting the model's estimated random-effect variance.

Calling `normalize_etas()` again merges into (rather than replacing) any previously-set factors, via `utils::modifyList()` – so `...` can be used to (re)compute just a subset of etas, eg after refitting. To turn normalization off again, assign directly: `xpdb$normalize_etas$ETA1 <- NULL` for a single eta, or `xpdb$normalize_etas <- NULL` for all of them.

Value

`xp_xtras` object, with the computed factors set under `$normalize_etas` (not `$options` – see Details)

See Also

`recalc_shk()`, which uses the same omega-matching logic

Examples

```
xpdb_norm <- normalize_etas(xpdb_x)
eta_grid(xpdb_norm)

# Just a subset of etas...
normalize_etas(xpdb_x, ETA1)

# By empirical SD instead, eg if the omega match fails or is unreliable
normalize_etas(xpdb_x, .use_sd = TRUE)
```

---

patch_condn *Patch condition number extraction*

Description

[Experimental]

Bugfix for `xpose:::sum_condn`, the internal function `xpose` uses to populate the `'condn'` (condition number) entry of `xpdb$summary`.

For NONMEM runs with more than one estimation method (e.g. `SAEM` followed by importance sampling), the `.lst` file contains more than one `EIGENVALUES OF COR MATRIX OF ESTIMATE` block. `xpose` always uses the *first* block found, which is not necessarily from the final estimation method, so the reported condition number can be wrong. This patch instead uses the *last* block, matching the value reported by NONMEM-adjacent tools such as PsN's `sumo`.

`xpose:::sum_condn()` itself (not this function) is also where a multi- method run with more than one `EIGENVALUES OF COR` block raises "numerical expression has ... elements: only the first used" – it runs automatically inside `xpose::xpose_data()`, before `patch_condn()` gets a chance to run, so that warning is expected and cannot be suppressed from here; this function only fixes the resulting `'condn'` value afterward.

Usage

```
patch_condn(xpdb)
```

Arguments

xpdb	An `xpose_data` or `xp_xtras` object.

Value

The `xpdb` object, with a corrected `'condn'` entry in `xpdb$summary` (unchanged if `xpdb` is not from `nonmem`, or if no eigenvalues could be found).

Examples

```
xpdb_ex_pk <- patch_condn(xpose::xpdb_ex_pk)
```

---

persist_process_presets

Write current process presets out to a `.Rprofile`

Description

Syncs the in-session `add_process_preset()` registry out to a `.Rprofile` file, so it's available again in future sessions. This is what `persist = TRUE` on `add_process_preset()`/`remove_process_preset()`/`amend_process_preset()` calls internally; call it directly to persist several in-session changes (each made with `persist = FALSE`) in one write/confirmation instead of one per change. See the CRAN-policy notes in `add_process_preset()` – in particular, this always errors instead of writing anything when `rlang::is_interactive()` is `FALSE`.

Usage

```
persist_process_presets(ask = TRUE, profile = "project")
```

Arguments

ask	<logical(1)> When persist = TRUE, ask for interactive confirmation before writing? (default: TRUE; only ever consulted when rlang::is_interactive() is already TRUE)
profile	<character(1)> Where to persist to when persist = TRUE: "project" (default, .Rprofile in getwd()), "user" (the user-level profile), or a literal file path.

Value

TRUE if the file was written, FALSE if declined (invisibly)

Examples

```
## Not run:
add_process_preset(~ .x %>% as_xpdb_x(), name = "convert", persist = FALSE)
persist_process_presets()

## End(Not run)
```

---

pheno_base *An* xp_xtras *example of a base model*

Description

Base model for phenobarbital in neonates.

Usage

```
pheno_base
```

Format

xp_xtras:
An xp_xtras object.

Details

This is run6 in <pheno_set>

Source

doi:10.1159/000457062 and nlmixr2data::pheno_sd

pheno_final *An* xp_xtras *example of a final model*

Description

Final model for phenobarbital in neonates.

Usage

```
pheno_final
```

Format

xp_xtras:
An xp_xtras object.

Details

This is re-parameterized from the covariate-building work, which in this case did not identify a relationship with Apgar score.

This is run16 in <pheno_set>

Source

doi:10.1159/000457062 and nlmixr2data::pheno_sd

pheno_saem *An* xp_xtras *example of a final model*

Description

Final model for phenobarbital in neonates.

Usage

```
pheno_saem
```

Format

xp_xtras:
An xp_xtras object.

Details

This is the same as pheno_final but fitted with SAEM/IMP.

Not a part of <pheno_set>

Source

doi:10.1159/000457062 and nlmixr2data::pheno_sd

---

pheno_set *A more complex example of* xpose_set *object*

Description

Model-building set for the phenobarbital in neonates PK data used across multiple packages.

Usage

```
pheno_set
```

Format

xpose_set:
An xpose_set object of length 14 with a branched lineage.

Details

This is not a demonstration of high-quality model-building, it is just a typical and simple example.

Source

doi:10.1159/000457062 and nlmixr2data::pheno_sd

---

pkpd_m3 *An* xp_xtras *example of an M3 model*

Description

A representative PK/PD model with M3 fitting applied.

Usage

```
pkpd_m3
```

Format

xp_xtras:
An xp_xtras object.

Source

doi:10.1002/psp4.13219

References

Beal, S.L. Ways to Fit a PK Model with Some Data Below the Quantification Limit. J Pharmacokinet Pharmacodyn 28, 481-504 (2001). doi:10.1023/A:1012299115260

Prybylski JP. Indirect modeling of derived outcomes: Are minor prediction discrepancies a cause for concern? CPT Pharmacometrics Syst Pharmacol. 2024; 00: 1-9. doi:10.1002/psp4.13219

Examples

```
# To establish as a complete categorical DV example:
pkpd_m3 <- pkpd_m3 %>%
  # Need to ensure var types are set
  set_var_types(catdv=BLQ,dvprobs=LIKE) %>%
  # Set probs
  set_dv_probs(1, 1~LIKE, .dv_var = BLQ) %>%
  # Optional, but useful to set levels
  set_var_levels(1, BLQ = lvl_bin())
```

---

pkpd_m3_df *An* xp_xtras *example of an M3 model (dataset)*

Description

The dataset used to fit the pkpd_m3 model.

Usage

```
pkpd_m3_df
```

Format

xp_xtras:

An xp_xtras object.

Source

doi:10.1002/psp4.13219

References

Prybylski JP. Indirect modeling of derived outcomes: Are minor prediction discrepancies a cause for concern? CPT Pharmacometrics Syst Pharmacol. 2024; 00: 1-9. doi:10.1002/psp4.13219

plot.xpose_data *Generate a batch of diagnostic plots from an* xpdb

Description

Runs a list of plot-generating calls against `x` in one go, returning a flat, named list of the results. Each element of `plots` (or the resolved default, see Details) is either:

- a bare function taking a single xpdb argument, or
- a one-sided formula in the `~ fn(.x)` idiom used elsewhere in this package (see `focus_function()`), letting extra arguments be baked straight into the call – e.g. `~ xpose::res_vs_idv(.x, res = "CWRES")`.

Both forms are converted to callables with `purrr::as_mapper()`. Because entries are just calls, the same underlying plot function can appear more than once with different options (e.g. `res_vs_idv` for both CWRES and IWRES) – see the examples.

If a single entry itself returns a list of plots (as e.g. `eta_grid()` can, given more than one eta), that list is flattened into the overall output rather than kept as a nested list – so the return value is always a flat list of ggplot/xpose_plot objects.

Usage

```
## S3 method for class 'xpose_data'
plot(x, y, plots, ..., force = FALSE, quiet)
```

Arguments

x	<xpose_data> or <xp_xtras> object
y	unused; present only for consistency with the `graphics::plot()` generic
plots	A list of plot specs (see Description); defaults to the resolved value described in Details
...	unused
force	<logical> If FALSE (the default), a failing plot immediately aborts the whole call. If TRUE, a failing plot is instead skipped (with a warning), and the rest of `plots` is still attempted.
quiet	<logical> Silence the loading spinner and the summary warning issued when `force = TRUE` and at least one plot failed; defaults to `x$options$quiet`

Details

`plots` is resolved, in increasing precedence, from: this package's built-in default (`dv_vs_ipred`, `dv_vs_pred`, `res_vs_idv`, `res_vs_pred`, `eta_distrib`, `eta_grid`, `eta_vs_cov_grid`, `ind_plots_sample`), the `xpose.xtras.default_plots` session option, an xpdb-level default set via `set_default_plots()`, and finally the `plots` argument itself, if supplied.

A loading spinner is shown (interactive sessions only, unless `quiet = TRUE`) while plots are generated. If a plot fails to generate, the default (`force = FALSE`) is to immediately raise an error (showing

the original error as its parent) without returning any plots at all. With `force = TRUE`, a failure is instead emitted as a warning and that entry is skipped, so the rest of `plots` still gets a chance to run.

Value

A flat, named list of ggplot/xpose_plot objects

Examples

```
# the package's built-in default battery of diagnostic plots
default_plots <- plot(xpdb_x, quiet = TRUE)
names(default_plots)

# a custom spec: bare functions, formulas, and the same function twice
custom_plots <- plot(
  xpdb_x,
  plots = list(
    xpose::dv_vs_ipred,
    ~ xpose::res_vs_idv(.x, res = "CWRES"),
    ~ xpose::res_vs_idv(.x, res = "IWRES")
  ),
  quiet = TRUE
)
names(custom_plots)
```

---

prm_contcov *Continuous/categorical covariate effect tables*

Description

Computes, for each covariate association declared with `add_cov_association()`, the covariate's effect on the associated parameter (as a ratio to the parameter's typical value, 1 at the reference covariate value/level) at a handful of representative evaluation points, with an uncertainty interval propagated from the effect-size theta's standard error. This is what `xplot_forest()` plots; calling these directly is mostly useful for inspecting the numbers before/without plotting.

`prm_contcov()` handles continuous covariates (`linear`, `power`, `exponential`, `hockey`, or `custom` associations), evaluated at the low/reference/high points. `prm_catcov()` handles categorical covariates (`catshift` or `custom` associations), evaluated at every observed level. `prm_cov()` combines both.

Usage

```
prm_contcov(
  xpdb,
  ...,
  .problem = NULL,
  .subprob = NULL,
```

```
  .method = NULL,
  ci_method = c("simulation", "delta"),
  probs = c(0.05, 0.95),
  level = 0.95,
  nsim = 1000,
  keep_draws = FALSE,
  quiet
)

prm_catcov(
  xpdb,
  ...,
  .problem = NULL,
  .subprob = NULL,
  .method = NULL,
  ci_method = c("simulation", "delta"),
  level = 0.95,
  nsim = 1000,
  keep_draws = FALSE,
  quiet
)

prm_cov(
  xpdb,
  ...,
  .problem = NULL,
  .subprob = NULL,
  .method = NULL,
  ci_method = c("simulation", "delta"),
  probs = c(0.05, 0.95),
  level = 0.95,
  nsim = 1000,
  keep_draws = FALSE,
  quiet
)
```

Arguments

xpdb	<xp_xtras> object with associations declared via add_cov_association()
...	<dynamic-dots> Optional param ~ covariate selectors (bare, unquoted, same style as drop_cov_association()) to restrict which declared associations are computed. Defaults to all of them.
.problem	<numeric> Problem number.
.subprob	<numeric> Subprob number.
.method	<numeric> Method.
ci_method	<character> "simulation" (default) draws nsim samples of each theta from N(theta_hat, se) (mirrors mutate_prm()'s .autose approach) and propagates them through the (possibly nonlinear) effect_ratio function, taking the

	resulting sample quantiles as the interval; most accurate for strongly nonlinear forms (power, exponential, hockey). "delta" is a first-order analytic (numerical-gradient) log-scale approximation – cheap and deterministic, but less accurate the more nonlinear the association is. Both treat multiple thetas (eg hockey, multi-level catshift) as independent, ignoring any covariance between them.
probs	<numeric(2)> For prm_contcov(): quantiles of the covariate's observed data used as the "low"/"high" evaluation points.
level	<numeric> Confidence level for the effect interval.
nsim	<numeric> Number of simulation draws, when ci_method = "simulation".
keep_draws	<logical> If TRUE (requires ci_method = "simulation"), attach a draws list-column: the raw nsim simulated effect-ratio draws behind each row's CI. Mainly intended for a forest-plot violin/density layer; most users won't need this.
quiet	Silence extra output.

Value

A prm_cov_tbl tibble with one row per (parameter, covariate, evaluation point): param, covariate, covtype, level ("low"/"ref"/"high" for continuous, the raw category value for categorical), value (the covariate value/level backing that row), is_ref (TRUE for the reference row/level – always effect/ci_low/ci_high == 1, by construction), effect, ci_low, ci_high, ci_method.

See Also

add_cov_association(), xplot_forest()

Examples

```
xpdb_x %>%
  add_cov_association(TVCL ~ power(CLCR, THETA7, ref = 64)) %>%
  prm_contcov()

xpdb_x %>%
  add_cov_association(TVCL ~ catshift(SEX, THETA4, ref = 1)) %>%
  prm_catcov()

xpdb_x %>%
  add_cov_association(
    TVCL ~ power(CLCR, THETA7, ref = 64),
    TVCL ~ catshift(SEX, THETA4, ref = 1)
  ) %>%
  prm_cov()

# Restrict to one association, and use the analytic delta-method CI
xpdb_x %>%
  add_cov_association(
    TVCL ~ power(CLCR, THETA7, ref = 64),
    TVCL ~ catshift(SEX, THETA4, ref = 1)
  ) %>%
```

```
prm_cov(TVCL ~ CLCR, ci_method = "delta")
```

`prm_waterfall` *Specific waterfall plots*

Description

Differences are second listed model minus first listed. Eg, in `eta_waterfall(run1,run2)`, the when etas in run2 are greater than those in run1, the difference will be positive.

Usage

```
prm_waterfall(
  xpdb_s,
  ...,
  .inorder = FALSE,
  type = "bh",
  max_nind = 0.7,
  scale_diff = TRUE,
  show_n = TRUE,
  title = "Parameter changes between models | @run",
  subtitle = "Based on @nobs observations in @nind individuals",
  caption = "@dir",
  tag = NULL,
  facets = NULL,
  facet_scales = "free_x",
  .problem,
  .subprob,
  .method,
  quiet
)

eta_waterfall(
  xpdb_s,
  ...,
  .inorder = FALSE,
  type = "bh",
  max_nind = 0.7,
  scale_diff = TRUE,
  show_n = TRUE,
  title = "Eta changes between models | @run",
  subtitle = "Based on @nobs observations in @nind individuals",
  caption = "@dir",
  tag = NULL,
  facets = NULL,
  facet_scales = "free_x",
```

```
  .problem,
  .subprob,
  .method,
  quiet
)

iofv_waterfall(
  xpdb_s,
  ...,
  .inorder = FALSE,
  type = "bh",
  max_nind = 0.7,
  scale_diff = FALSE,
  show_n = TRUE,
  title = "iOfv changes between models | @run",
  subtitle = "Based on @nobs observations in @nind individuals",
  caption = "@dir",
  tag = NULL,
  facets = NULL,
  facet_scales = "free_x",
  .problem,
  .subprob,
  .method,
  quiet
)
```

Arguments

xpdb_s	<xpose_set> object
...	See <two_set_dots>
.inorder	See <two_set_dots>
type	See Details.
max_nind	If less than 1, the percentile of absolute change values above which to plot. If above 1, the absolute number of subjects is included. To show all, use an extreme positive number like 9999.
scale_diff	<logical> Scale change to the standard deviation of the model 1 column values. Respects faceting.
show_n	<logical> For faceting variables, show N per facet. *Not implemented*
title	Plot title
subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag
facets	<character> Faceting variables
facet_scales	<character> Forwarded to facet_*(scales = facet_scales)
.problem	The problem to be used, by default returns the last one.

.subprob	The subproblem to be used, by default returns the last one.
.method	The estimation method to be used, by default returns the last one.
quiet	Silence extra debugging output

Details

For type-based customization of plots:

- b bar plot (from geom_bar)
- h hline at 0 (from geom_hline)
- t text of change value (from geom_text)

Value

<xpose_plot> object

Examples

```
# Parameter value changes
pheno_set %>%
  # Ensure param is set
  focus_qapply(set_var_types, param=c(CL,V)) %>%
  prm_waterfall(run5,run6)


# EBE value changes
pheno_set %>%
  eta_waterfall(run5,run6)

# iOFV changes
pheno_set %>%
  focus_qapply(backfill_iofv) %>%
  # Note the default scaling is flipped here
  iofv_waterfall(run5,run6)
```

---

recalc_shk *Recalculate eta shrinkage from individual estimates*

Description

Unlike get_shk(), which parses the eta shrinkage NONMEM itself reported in the output file, this recalculates shrinkage directly from the individual (empirical Bayes) eta estimates found in the data, using the standard $100 \times (1 - SD(\eta)/\omega)$ formula, where ω is the standard deviation implied by the associated diagonal omega estimate.

Usage

```
recalc_shk(
  xpdb,
  ...,
  .etastype = 1,
  .problem = NULL,
  .subprob = NULL,
  .method = NULL,
  drop_fixed = TRUE,
  quiet
)
```

Arguments

xpdb	<xpose_dataxpose::xpose_data> or xp_xtras object
...	<tidyselect> Which eta column(s) to recalculate shrinkage for. Defaults to every eta column for .problem.
.etastype	<numeric(1)> 1 (the default) excludes, for each eta, individuals whose estimate is a "true zero" (exactly 0, as opposed to merely shrunk near it) from the calculation; 0 keeps them. See Details.
.problem	<numeric> Problem number to use. Uses the xpose default if not provided.
.subprob	<numeric> Subproblem number to use. Uses the xpose default if not provided.
.method	<character> Method to use. Uses the xpose default if not provided.
drop_fixed	<logical> Drop fixed etas (which have no meaningful shrinkage to recalculate), as in xpose::drop_fixed_cols.
quiet	<logical> Silence extra debugging output

Details

An eta is a "true zero" for an individual when NONMEM never had grounds to move it away from its prior mean of 0, eg an individual with no observations contributing to the objective function. That is different from an eta that is merely shrunk close to 0 through legitimate estimation, and including "true zero" individuals in the shrinkage calculation biases it, since they carry no information about the actual empirical distribution of etas. .etastype = 1 (the default) excludes them from the calculation; .etastype = 0 reproduces the traditional, unadjusted calculation.

Value

A tibble with one row per eta, reporting the omega used, the number of individuals excluded (if any), and the recalculated shrinkage (as a percentage, to stay consistent with get_shk()).

Examples

```
recalc_shk(xpdb_x)

# Just a subset of etas...
recalc_shk(xpdb_x, ETA1)
```

```
# Including "true zero" etas in the calculation
recalc_shk(xpdb_x, .etastype = 0)
```

`reportable_digits` *Reportable digits for model fit*

Description

An opinionated function where for optimization routines that report number of significant digits (eg, FO-based), only those number of digits are considered reportable.

Usage

```
reportable_digits(xpdb, .default = 3, .problem, .subprob, .method)
```

Arguments

`xpdb`	<xpose_dataxpose::xpose_data> object
`.default`	<numeric> Default number of digits to return if not found
`.problem`	<numeric> Problem number to use. Uses all problem if not provided.
`.subprob`	<numeric> Subproblem number to use. Uses the xpose default if not provided.
`.method`	<character> Method to use. Uses the xpose default if not provided.

Value

Number of reportable digits

Examples

```
reportable_digits(xpdb_x)
```

`reshape_set` *Convert xpose_set to a nested list.*

Description

This amounts to a convenience function for tidy manipulations.

Usage

```
reshape_set(x)

unreshape_set(y)
```

Arguments

x	<xpose_set> An xpose_set object
y	<tibble> A nested table from an xpose_set

Value

<tibble> Nested list, or <xpose_set>

Examples

```
rset <- reshape_set(xpdb_set)
# Properties (exposed and top-level) can be seen. xpdb objects are nested in the xpdb column.
rset %>% dplyr::select(-xpdb) %>% dplyr::glimpse()

unreshape_set(rset)

# The reversibility of reshaping can be confirmed:
identical(xpdb_set,reshape_set(xpdb_set) %>% unreshape_set())
```

---

roc_by_mod *ROC curve across models*

Description

Faceted display of ROC curves across models in a set.

Usage

```
roc_by_mod(
  xpdb_s,
  ...,
  .lineage = FALSE,
  mapping = NULL,
  cutpoint = 1,
  type = "ca",
  title = "ROC curves across models | @dvcol~@probcol",
  subtitle = "Based on @nind individuals, Ofvs: @ofv",
  caption = "@dir",
  tag = NULL,
  axis.text = "@run",
  facets,
  .problem,
  quiet,
  roc_args = NULL
)
```

Arguments

xpdb_s	<xpose_set> object
...	Any additional aesthetics.
.lineage	<logical> where if TRUE, ... is processed
mapping	ggplot2 style mapping
cutpoint	<numeric> Of defined probabilities, which one to use in plots.
type	See Details.
title	Plot title
subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag
axis.text	What to label the model. This is parsed on a per-model basis.
facets	Additional facets
.problem	Problem number
quiet	Silence extra debugging output
roc_args	Additional arguments to pass to xplot_rocplot()

Details

For type-based customization of plots:

- c ROC curve (using geom_path)
- k Key points on ROC curve (where on curve the threshold is thres_fixed) (using geom_point)
- p ROC space points (using geom_point)
- t ROC space text (using geom_text)
- a AUC in bottom right (using geom_label)

Examples

```
pkpd_m3 <- pkpd_m3 %>%
  # Need to ensure var types are set
  set_var_types(catdv=BLQ,dvprobs=LIKE) %>%
  # Set probs
  set_dv_probs(1, 1~LIKE, .dv_var = BLQ) %>%
  # Optional, but useful to set levels
  set_var_levels(1, BLQ = lvl_bin())

m3_set <- xpose_set(
  run1=set_prop(pkpd_m3,run="run1"),
  run2=set_prop(pkpd_m3,run="run2"),
  run3=set_prop(pkpd_m3,run="run3")
)

roc_by_mod(m3_set, type = "ck", quiet = TRUE)
```

roc_plot *ROC Plot for categorical DVs*

Description

ROC Plot for categorical DVs

Usage

```
roc_plot(
  xpdb,
  mapping = NULL,
  cutpoint = 1,
  group = "ID",
  type = "ca",
  title = "ROC curve @dvcol~@probcol | @run",
  subtitle = "Ofv: @ofv, Eps shrink: @epsshk",
  caption = "@dir",
  tag = NULL,
  guide = TRUE,
  facets,
  .problem,
  quiet,
  ...
)
```

Arguments

xpdb	<xp_xtras> or <xpose_data> object
mapping	ggplot2 style mapping
cutpoint	<numeric> Of defined probabilities, which one to use in plots.
group	Variable by which to group points or text
type	See Details.
title	Plot title
subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag
guide	Include unity line?
facets	Additional facets
.problem	Problem number
quiet	Silence extra debugging output
...	Any additional aesthetics.

Details

For type-based customization of plots:

- c ROC curve (using geom_path)
- k Key points on ROC curve (where on curve the threshold is thres_fixed) (using geom_point)
- p ROC space points (using geom_point)
- t ROC space text (using geom_text)
- a AUC in bottom right (using geom_label)

Value

A desired plot

See Also

catdv_vs_dvprobs()

Examples

```
# Note these examples are similar to catdv_vs_dvprobs

## Not run:
# Test M3 model
pkpd_m3 %>%
  # Need to ensure var types are set
  set_var_types(catdv=BLQ,dvprobs=LIKE) %>%
  # Set probs
  set_dv_probs(1, 1~LIKE, .dv_var = BLQ) %>%
  # Optional, but useful to set levels
  set_var_levels(1, BLQ = lvl_bin()) %>%
  # Generate typical ROC curve
  roc_plot()

# Test categorical model
vismo_xpdb <- vismo_pomod  %>%
  set_var_types(.problem=1, catdv=DV, dvprobs=matches("^P\\d+$")) %>%
  set_dv_probs(.problem=1, 0~P0,1~P1,ge(2)~P23)

# Various cutpoints (note axes labels and texts)
vismo_xpdb %>%
  roc_plot(type = "p") # space plot
vismo_xpdb %>%
  roc_plot(cutpoint=2, type = "cak") # with area and key point
vismo_xpdb %>%
  roc_plot(cutpoint=3, type = "cak")

# alternative model example
vismo_xpdb2  <- vismo_dtmm   %>%
  set_var_types(.problem=1, catdv=DV, dvprobs=matches("^P\\d+$")) %>%
  set_dv_probs(.problem=1, 0~P0,1~P1,ge(2)~P23)
```

```
vismo_xpdb2 %>%
  roc_plot(cutpoint=2, type = "cak")

## End(Not run)
```

`set_base_model` *Base model for* `xpose_set`

Description

Base model for `xpose_set`

Usage

```
set_base_model(xpdb_s, ...)

get_base_model(xpdb_s)

unset_base_model(xpdb_s)
```

Arguments

`xpdb_s`	<xpose_set> object
`...`	«dynamic-dots» name of base model

Value

<xpose_set> object with a base model

Examples

```
w_base <- xpdb_set %>%
  set_base_model(mod2)
w_base # base model listed in output

get_base_model(w_base) # base model name

unset_base_model(w_base) # base model no longer in output
```

set_default_labs *Set default plot label overrides on an* xp_xtras *object*

Description

Stores title/subtitle/caption/tag templates on xpdb that apply_default_labs() will use to fill in (or overwrite) labels on any plot built from this xpdb, when xpdb is passed to it. Values may contain the same @keyword placeholders understood by xpose::parse_title() (e.g. "@nind", "@nobs", "@run"), since they are resolved the same way.

Usage

```
set_default_labs(xpdb, ...)
```

Arguments

xpdb	<xpose_data> or <xp_xtras> object
...	<dynamic-dots> One or more of title, subtitle, caption, tag, given as character strings

Value

xp_xtras object

See Also

set_xtras_options() for the session-option equivalent (xpose.xtras.default_labs), and get_xtras_option() to check which one is currently dominant.

Examples

```
xpdb_x <- set_default_labs(xpdb_x, caption = "@nobs observations in @nind individuals")
p <- xpose::dv_vs_ipred(xpdb_x)
apply_default_labs(p, xpdb = xpdb_x)
```

set_default_plots *Set a default plot spec on an* xp_xtras *object*

Description

Stores a plot spec on xpdb that plot.xpose_data() uses instead of the package's built-in default whenever plots isn't supplied directly to that call. Unlike set_default_labs()/set_default_watermark(), this replaces the xpdb-level default wholesale rather than merging with it: entries in a plot spec aren't addressable by a stable key (the same plot function can legitimately appear more than once), so there is no sensible key-by-key merge to perform.

Usage

```
set_default_plots(xpdb, plots)
```

Arguments

xpdb	<xpose_data> or <xp_xtras> object
plots	A list of plot specs – see plot.xpose_data() for the accepted forms

Value

xp_xtras object

See Also

set_xtras_options() for the session-option equivalent (xpose.xtras.default_plots), and get_xtras_option() to check which one is currently dominant.

Examples

```
xpdb2 <- set_default_plots(xpdb_x, list(~ xpose::dv_vs_ipred(.x), ~ xpose::eta_distrib(.x)))
names(plot(xpdb2, quiet = TRUE))
```

---

set_default_watermark *Set default watermark options on an* xp_xtras *object*

Description

Stores default add_watermark() arguments on xpdb that add_watermark() will use for any of label/colour/alpha/size/ angle/fontface not otherwise supplied, when xpdb is passed to it.

Usage

```
set_default_watermark(xpdb, ...)
```

Arguments

xpdb	<xpose_data> or <xp_xtras> object
...	<dynamic-dots> One or more of label, colour, alpha, size, angle, fontface

Value

xp_xtras object

See Also

set_xtras_options() for the session-option equivalent (xpose.xtras.default_watermark), and get_xtras_option() to check which one is currently dominant.

Examples

```
xpdb_x <- set_default_watermark(xpdb_x, label = "PRELIMINARY", colour = "red")
p <- xpose::dv_vs_ipred(xpdb_x)
add_watermark(p, xpdb = xpdb_x)
```

`set_dv_probs` *Set probability columns for categorical endpoints*

Description

For categorical DVs or similar endpoints (such as censoring flag columns, like `BLQ`), this function allows probability columns to be defined for each level.

Usage

```
set_dv_probs(
  xpdb,
  .problem = NULL,
  ...,
  .dv_var = NULL,
  .handle_missing = c("quiet", "warn", "error")
)
```

Arguments

`xpdb`	`<xp_xtras>` object
`.problem`	`<numeric>` Problem number to use. Uses all problems if `NULL` (the default). May be omitted entirely and left to default, even when formulas are supplied positionally in `....`
`...`	Formulas where LHS are levels or pseudo-functions (see Details), and RHS are columns with probabilities of those levels.
`.dv_var`	`<tidyselect>` of column having the categorical observation. Default is first-listed `catdv`.
`.handle_missing`	`<character>` How to handle missing levels: "quiet", "warn", or "error"

Details

The same probability cannot be assigned to multiple values. Pseudo-functions can be used, or new columns can be created to overcome this limitation. The available pseudo-functions should be written like `ge(value)` (for >=), `gt(value)` (for >), etc. These comparison names are those used in Perl, Fortran and many other languages. The function `eq()` should not be used, but it will be ignored either way; equivalence is implied with the base syntax.

Value

`<xp_xtras>` object with updated probabilities

Examples

```
pkpd_m3 %>%
 # Not necessary, but correct to set var type before using this
 set_var_types(.problem=1, catdv=BLQ, dvprobs=LIKE) %>%
 # Set var type. Warnings can be helpful unless an inverse likelihood column is available
 set_dv_probs(.problem=1, 1~LIKE, .dv_var = BLQ, .handle_missing = "warn") %>%
 list_vars()

# Same as above with demo of inverse column
pkpd_m3 %>%
 xpose::mutate(INVLIKE = 1-LIKE) %>%
 set_var_types(.problem=1, catdv=BLQ, dvprobs=c(LIKE,INVLIKE)) %>%
 # Note no warning
 set_dv_probs(.problem=1, 1~LIKE, 0~INVLIKE, .dv_var = BLQ, .handle_missing = "warn")%>%
 list_vars()

# With categorical model
vismo_pomod  %>%
 # Update var types
 set_var_types(.problem=1, catdv=DV, dvprobs=matches("^P\\d+$")) %>%
 # Warning (as noted), does not recognize 3 is covered implicitly. That's ok!
 set_dv_probs(.problem=1, 0~P0,1~P1,ge(2)~P23, .handle_missing = "warn")%>%
 list_vars()

# Same as above, but...
vismo_pomod  %>%
 set_var_types(.problem=1, catdv=DV, dvprobs=matches("^P\\d+$")) %>%
 # Default is to not bother users with a warning
 set_dv_probs(.problem=1, 0~P0,1~P1,ge(2)~P23)%>%
 list_vars()

# .problem can be omitted for single-problem models
pkpd_m3 %>%
 set_var_types(catdv=BLQ, dvprobs=LIKE) %>%
 set_dv_probs(1~LIKE, .dv_var = BLQ) %>%
 list_vars()
```

`set_option` *Set an* xpose *option*

Description

Sets one or more entries in `xpdb$options`, merged in via `utils::modifyList()` – which recurses into list-valued options, so setting a single key of an existing named-list option (e.g. one label of `default_labs`, see `set_default_labs()`) leaves its other keys untouched rather than replacing the whole list. This is what `set_default_labs()` and `set_default_watermark()` are built on, and calling `set_option()` directly with `default_labs`/`default_watermark` behaves the same way.

Usage

```
set_option(xpdb, ...)
```

Arguments

xpdb	<xpose_dataxpose::xpose_data> object
...	<dynamic-dots> Arguments in the form of option = value

Value

xp_xtras object

Examples

```
xpdb_x <- set_option(xpdb_x, quiet = TRUE)
```

---

set_prop *Set a summary property*

Description

Set a summary property

Usage

```
set_prop(xpdb, ..., .problem = NULL, .subprob = NULL)
```

Arguments

xpdb	<xpose_dataxpose::xpose_data> object
...	<dynamic-dots> defining which properties to transform. Argument should be valid label.
.problem	<numeric> Problem number to use. Uses all problem if not provided.
.subprob	<numeric> Subproblem number to use. Uses the xpose default if not provided.

Details

Although one might be tempted to set custom properties using this function, with the intention to maintain cross-functionality with xpose, users cannot set a non-existent property with this function. When used internally, workarounds to this semi-limitation are used.

Value

xp_xtras object

Examples

```
set_prop(xpose::xpdb_ex_pk, descr = "New model description") %>%
  xpose::get_summary()
```

`set_var_levels` *Set variable levels*

Description

For variable types such as `catcov`, it can be convenient to define levels. This function provides a straightforward means to do so, consistent with `tidy` functions like <`case_when`>.

Several convenience functions are provided for common levels in <`levelers`>.

Usage

```
set_var_levels(
  xpdb,
  .problem = NULL,
  ...,
  .missing = "Other",
  .ordered = character(),
  .handle_missing = c("quiet", "warn", "error")
)
```

Arguments

`xpdb`	<`xp_xtras`> object
`.problem`	<`numeric`> Problem number to use. Uses the all problems if `NULL`
`...`	<`list`> of formulas or leveler functions, where the relevant variable is provided as the argument,
`.missing`	<`character`> Value to use for missing levels
`.ordered`	<`character`> Names of columns whose levels should be treated as an ordered factor (see `base::factor`), even when supplied as a plain formula list rather than via `lvl_inord()`. Columns leveled with `lvl_inord()` are already ordered by default and do not need to be listed here.
`.handle_missing`	<`character`> How to handle missing levels: "quiet", "warn", or "error"

Value

<`xp_xtras`> object with updated levels

Examples

```
set_var_levels(xpdb_x,
  SEX = lvl_sex(),
  MED1 = lvl_bin(),
  MED2 = c(
    0 ~ "n",
    1 ~ "y"
  )
)
```

`set_var_types` *Set variable types*

Description

[Experimental]

<set_var_types> wrapper that accepts tidyselect syntax. Character vector-based selection still works.

`set_var_types_x` accepts `xpose_data` or `xp_xtras` objects.

`set_var_types` without `_x` is defined with S3 methods. To maintain `xpose` expectations, the default method is <set_var_types>, but if an `xp_xtras` object is used, the method uses `set_var_types_x`.

Usage

```
set_var_types(xpdb, .problem = NULL, ..., auto_factor = TRUE, quiet)
```

Arguments

`xpdb`	An `xpose_data` object.
`.problem`	The problem number to which the edits will be applied.
`...`	<dynamic-dots> Passed to <set_var_types> after processing.
`auto_factor`	If `TRUE` new columns assigned to the type 'catcov' will be converted to factor.
`quiet`	Logical, if `FALSE` messages are printed to the console.

Value

An xpose_data object

Examples

```
data("xpdb_ex_pk", package = "xpose")

# Change variable type
xpdb_2 <- set_var_types_x(
  xpdb_ex_pk, .problem = 1,
  idv = TAD,
  catcov = starts_with("MED"),
  contcov = c(CLCR,AGE)
  )
```

`set_var_types.default` *Set variable types*

Description

[Experimental]

<set_var_types> wrapper that accepts tidyselect syntax. Character vector-based selection still works.

`set_var_types_x` accepts `xpose_data` or `xp_xtras` objects.

`set_var_types` without `_x` is defined with S3 methods. To maintain `xpose` expectations, the default method is <set_var_types>, but if an `xp_xtras` object is used, the method uses `set_var_types_x`.

Usage

```
## Default S3 method:
set_var_types(xpdb, .problem = NULL, ..., auto_factor = TRUE, quiet)
```

Arguments

`xpdb`	An `xpose_data` object.
`.problem`	The problem number to which the edits will be applied.
`...`	<dynamic-dots> Passed to <set_var_types> after processing.
`auto_factor`	If `TRUE` new columns assigned to the type 'catcov' will be converted to factor.
`quiet`	Logical, if `FALSE` messages are printed to the console.

Value

An xpose_data object

Examples

```
data("xpdb_ex_pk", package = "xpose")

# Change variable type
xpdb_2 <- set_var_types_x(
  xpdb_ex_pk, .problem = 1,
  idv = TAD,
  catcov = starts_with("MED"),
  contcov = c(CLCR,AGE)
  )
```

`set_var_types.xp_xtras` *Set variable types*

Description

[Experimental]

<set_var_types> wrapper that accepts tidyselect syntax. Character vector-based selection still works.

`set_var_types_x` accepts `xpose_data` or `xp_xtras` objects.

`set_var_types` without `_x` is defined with S3 methods. To maintain `xpose` expectations, the default method is <set_var_types>, but if an `xp_xtras` object is used, the method uses `set_var_types_x`.

Usage

```
## S3 method for class 'xp_xtras'
set_var_types(xpdb, .problem = NULL, ..., auto_factor = TRUE, quiet)
```

Arguments

`xpdb`	An `xpose_data` object.
`.problem`	The problem number to which the edits will be applied.
`...`	<dynamic-dots> Passed to <set_var_types> after processing.
`auto_factor`	If `TRUE` new columns assigned to the type 'catcov' will be converted to factor.
`quiet`	Logical, if `FALSE` messages are printed to the console.

Value

An xpose_data object

Examples

```
data("xpdb_ex_pk", package = "xpose")

# Change variable type
xpdb_2 <- set_var_types_x(
  xpdb_ex_pk, .problem = 1,
  idv = TAD,
  catcov = starts_with("MED"),
  contcov = c(CLCR,AGE)
  )
```

`set_var_types_x` *Set variable types*

Description

[Experimental]

<set_var_types> wrapper that accepts tidyselect syntax. Character vector-based selection still works.

`set_var_types_x` accepts `xpose_data` or `xp_xtras` objects.

`set_var_types` without `_x` is defined with S3 methods. To maintain `xpose` expectations, the default method is <set_var_types>, but if an `xp_xtras` object is used, the method uses `set_var_types_x`.

Usage

```
set_var_types_x(xpdb, .problem = NULL, ..., auto_factor = TRUE, quiet)
```

Arguments

`xpdb`	An `xpose_data` object.
`.problem`	The problem number to which the edits will be applied.
`...`	<dynamic-dots> Passed to <set_var_types> after processing.
`auto_factor`	If `TRUE` new columns assigned to the type 'catcov' will be converted to factor.
`quiet`	Logical, if `FALSE` messages are printed to the console.

Value

An xpose_data object

Examples

```
data("xpdb_ex_pk", package = "xpose")

# Change variable type
xpdb_2 <- set_var_types_x(
  xpdb_ex_pk, .problem = 1,
  idv = TAD,
  catcov = starts_with("MED"),
  contcov = c(CLCR,AGE)
  )
```

`set_xtras_options` *Set* `xpose.xtras` *session options*

Description

Convenience wrapper around base `options()` for the `xpose.xtras.*` family of options recognized by this package. Prefixes are added automatically and names are validated against the list below, so e.g. `set_xtras_options(save_dir = "figures")` is equivalent to (but safer against typos than) `options(xpose.xtras.save_dir = "figures")`. Current values can be read back with regular `getOption()` (e.g. `getOption("xpose.xtras.save_dir")`), or with `get_xtras_option()` for `default_labs`/`default_watermark`, which also considers any xpdb-level default.

Usage

```
set_xtras_options(...)
```

Arguments

`...`	<dynamic-dots> One or more of `default_labs`, `default_watermark`, `save_dir`, `save_width`, `save_height`, `save_fun`, `gg_theme`, `xp_theme`, given as `name = value`

Details

Recognized options (all unset, i.e. `NULL`, by default, except `auto_apply` which defaults to `TRUE`):

`auto_apply` Whether print.xpose_plot() and `ggsave_xp()` automatically apply configured `default_labs`/`default_watermark` (labels always; a watermark only if `default_watermark` is actually set at some tier – there's no unprompted default watermark). Defaults to `TRUE`, but is a no-op until `default_labs`/`default_watermark` are themselves configured, so leaving it at its default has no visible effect on its own; set it to `FALSE` to opt out of the auto-apply behavior everywhere at once (or pass `apply_labs`/`apply_watermark` to a specific `ggsave_xp()` call to opt out just there). `print.xpose_plot()` only ever sees the session-wide option tier (not an xpdb-level one) for the same reason noted under `default_labs` below.

default_labs Named list of default title/subtitle/ caption/tag templates (may contain @keyword placeholders, see xpose::parse_title()). Used by apply_default_labs() (and by extension ggsave_xp()) as the lowest-precedence source for any label a plot doesn't already have – xpdb-level defaults (see set_default_labs()) and arguments passed directly to apply_default_labs() both take precedence over this option.

default_watermark Named list of default add_watermark() arguments (any of label/colour/alpha/size/angle/ fontface). Used by add_watermark() as the lowest-precedence source – xpdb-level defaults (see set_default_watermark()) and arguments passed directly to add_watermark() both take precedence over this option.

default_plots A list of plot specs (see plot.xpose_data()) used whenever plots isn't supplied directly to plot.xpose_data(). Unlike default_labs/ default_watermark, this is resolved as a whole (not merged key by key) – xpdb-level defaults (see set_default_plots()) take precedence over this option, and a plots argument passed directly takes precedence over both.

save_dir, save_width, save_height Defaults for the path/width/height arguments of ggsave_xp(), used whenever those arguments aren't supplied explicitly.

save_fun Default save function for ggsave_xp() (itself defaulting to ggplot2::ggsave() when this is unset), for swapping in a drop-in alternative such as reportifyr::ggsave_with_metadata() project-wide instead of passing save_fun to every ggsave_xp() call.

gg_theme, xp_theme Default ggplot2 theme / xpose xp_theme (see xpose::update_themes()) applied to an xpose_data object the first time it's converted via as_xpdb_x(), so a project-wide look can be set once per session instead of calling xpose::update_themes() on every xpdb individually. Has no effect on xpdbs that are already xp_xtras objects.

Value

the previous values of the options that were set, invisibly (see options())

See Also

get_xtras_option() to inspect which tier is currently dominant for a two-tier option; apply_default_labs(), add_watermark(), ggsave_xp(), and as_xpdb_x(), which consume these options.

Examples

```
set_xtras_options(save_dir = "figures", default_watermark = list(label = "DRAFT"))
getOption("xpose.xtras.save_dir")
set_xtras_options(save_dir = NULL, default_watermark = NULL)
```

---

shark_colors *Change colors of shark plots*

Description

This changes the point and text color in the xp_theme of an xpose_data object.

Usage

```
shark_colors(
  xpdb,
  upcolor = xp_xtra_theme(base_on = xpdb$xp_theme)$sharkup_color,
  dncolor = xp_xtra_theme(base_on = xpdb$xp_theme)$sharkdn_color
)
```

Arguments

xpdb	<xpose_data> object
upcolor	Color for increasing dOFV
dncolor	Color for decreasing dOFV

Value

<xpose_data> object

See Also

shark_plot()

Examples

```
# Where this would fit in a particular workflow
xpose_set(pheno_base, pheno_final) %>%
  # forward functions affecting xpdb objects
  focus_xpdb(everything()) %>%
  # Add iOFVs
  focus_function(backfill_iofv) %>%
  # Change color of all xpdb xp_themes (though only the first one needs to change)
  focus_function(
  function(x) shark_colors(
      x,
      upcolor = "purple",
      dncolor = "green"
    )) %>%
  # See new plot
  shark_plot()
```

---

shark_plot *Individual contributions to dOFV*

Description

This is intended to match the overall behavior of dOFV.vs.id() in xpose4, within the framework of the xpose_set object.

dofv_vs_id is an alias of the function shark_plot, for recognition.

Usage

```
shark_plot(
  xpdb_s,
  ...,
  .inorder = FALSE,
  type = "plt",
  alpha = 0.05,
  df = "guess",
  text_cutoff = 0.8,
  title = "Individual contributions to dOfv | @run",
  subtitle = "Based on @nind individuals, Ofvs: @ofv",
  caption = "@dir",
  tag = NULL,
  ylab = "dOfv",
  xlab = "Number of individuals removed",
  opt,
  facets = NULL,
  .problem,
  .subprob,
  .method,
  quiet
)

dofv_vs_id(
  xpdb_s,
  ...,
  .inorder = FALSE,
  type = "plt",
  alpha = 0.05,
  df = "guess",
  text_cutoff = 0.8,
  title = "Individual contributions to dOfv | @run",
  subtitle = "Based on @nind individuals, Ofvs: @ofv",
  caption = "@dir",
  tag = NULL,
  ylab = "dOfv",
  xlab = "Number of individuals removed",
  opt,
  facets = NULL,
  .problem,
  .subprob,
  .method,
  quiet
)
```

Arguments

xpdb_s	<xpose_set> object

...	See <two_set_dots>
.inorder	See <two_set_dots>
type	See Details.
alpha	alpha for LRT
df	degrees of freedom for LRT. If "guess" (default), then use the difference in the number of unfixed parameters.
text_cutoff	If less than 1, the percentile of absolute individual dOFV values above which to show labels of IDs. If above 1, the absolute number of IDs to show. To show all, use an extreme positive number like 9999.
title	Plot title
subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag
ylab	y-axis label
xlab	x-axis label
opt	User-specified data options. Only some of these will be used.
facets	<character> vector selecting facets, or NULL (default).
.problem	The problem to be used, by default returns the last one.
.subprob	The subproblem to be used, by default returns the last one.
.method	The estimation method to be used, by default returns the last one.
quiet	Silence extra debugging output

Details

For type-based customization of plots:

- p points (using aesthetics for sharkup and sharkdn)
- l lines for dOFV (both total dOFV and significance are plotted)
- t text (using aesthetics for shkuptxt and shkdntxt)

In xpose4, users can control sig.drop, but this function uses alpha and df to determine the critical delta by the likelihood ratio test. It is acknowledged there are situations where this may not be valid, but it is suggested that df or alpha be adjusted to meet the desired sig.drop.

```
my_alpha <- 0.05
my_df <- 1.34 # fractional, perhaps to account for different IIVs

my_sigdrop <- -stats::qchisq(1-my_alpha, my_df)
my_sigdrop
#> [1] -4.633671
# Then use alpha=my_alpha, df=my_df in `shark_plot` call.
```

Value

<xpose_plot> object

See Also

shark_colors()

Examples

```
pheno_set %>%
  # Make sure set has iofv var types defined
  focus_xpdb(everything()) %>%
  focus_function(backfill_iofv) %>%
  # Pick two models or consistent with two_set_dots()
  shark_plot(run6,run11)

pheno_set %>%
  # As before
  focus_xpdb(everything()) %>%
  focus_function(backfill_iofv) %>%
  # Add indicator (or use established covariate)
  mutate(APGRtest = as.numeric(as.character(APGR))<5) %>%
  # Pick two models or consistent with two_set_dots()
  shark_plot(run6,run11, facets = "APGRtest")
```

---

shk_grid *Shrinkage contribution grid plots*

Description

These mirror eta_grid()/eta_vs_cov_grid(), but for the per-individual shrinkage contribution diagnostic (shk type columns, see derive_shk()/backfill_shk()) instead of the etas themselves.

Usage

```
shk_grid(
  xpdb,
  mapping = NULL,
  shkvar = NULL,
  drop_fixed = TRUE,
  title = "Shrinkage contribution correlations | @run",
  subtitle = "Based on @nind individuals",
  caption = "@dir",
  tag = NULL,
  pairs_opts,
  .problem,
  quiet,
  ...
)
```

```
shk_vs_cov_grid(
  xpdb,
  mapping = NULL,
  shkvar = NULL,
  cols = NULL,
  covvar = NULL,
  covtypes = c("cont", "cat"),
  show_n = TRUE,
  drop_fixed = TRUE,
  title = "Shrinkage contribution covariate correlations | @run",
  subtitle = "Based on @nind individuals",
  caption = "@dir",
  tag = NULL,
  shkcov = TRUE,
  pairs_opts,
  .problem,
  quiet,
  ...
)
```

Arguments

xpdb	<xp_xtras> or <xpose_data> object
mapping	ggplot2 style mapping
shkvar	tidyselect for shk variables
drop_fixed	As in xpose
title	Plot title
subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag
pairs_opts	List of arguments to pass to _opts. See <xplot_pairs>
.problem	Problem number
quiet	Silence extra debugging output
...	Passed to xplot_pairs
cols	tidyselect for covariates variables
covvar	For shk_vs_cov_grid only: an alias for cols (matching the covvar argument of shk_vs_contcov()/shk_vs_catcov()). If supplied (non-NULL), takes precedence over cols.
covtypes	Subset to specific covariate type?
show_n	Count the number of IDs in each category
shkcov	For shk_vs_cov_grid, shk are sorted after covariates to give an x orientation to covariate relationships.

Value

xp_tras_plot object

Examples

```
xpdb_shk <- backfill_shk(xpdb_x)
shk_grid(xpdb_shk)
shk_vs_cov_grid(xpdb_shk)
```

---

shk_vs_catcov *Shrinkage contribution versus categorical covariates*

Description

Mirrors eta_vs_catcov(), but for the per-individual shrinkage contribution diagnostic (shk type columns, see derive_shk()/backfill_shk()) instead of the etas themselves.

Usage

```
shk_vs_catcov(
  xpdb,
  mapping = NULL,
  shkvar = NULL,
  covvar = NULL,
  drop_fixed = TRUE,
  orientation = "x",
  show_n = check_xpdb_x(xpdb, .warn = FALSE),
  type = "bol",
  list = TRUE,
  title = "Shrinkage contribution versus categorical covariates | @run",
  subtitle = "Based on @nind individuals",
  caption = "@dir",
  tag = NULL,
  facets,
  .problem,
  quiet,
  ...
)
```

Arguments

xpdb	<xp_xtras> or <xpose_data> object
mapping	ggplot2 style mapping
shkvar	tidyselect for shk variables

covvar	tidyselect for categorical covariate variables; NULL (default) selects every categorical covariate in the xpdb data index.
drop_fixed	As in xpose
orientation	Passed to xplot_boxplot
show_n	Add "N=" to plot
type	Passed to xplot_boxplot
list	<logical> Only relevant when shkvar resolves to more than one shk column. If TRUE (default, for backwards compatibility), returns a plain list of one plot per shk column. If FALSE, they are instead combined onto one shared plot – faceted by shk column, in addition to the existing per-covariate facet – automatically paginating (at most 9 panels per page, i.e. ncol/nrow of 3) via xpose's own facet_wrap_paginate mechanism. Printing the returned plot renders every page; pass page to print() to select a specific one.
title	Plot title
subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag
facets	Additional facets
.problem	Problem number
quiet	Silence output
...	Any additional aesthetics.

Value

The desired plot, or (when shkvar resolves to more than one shk column and list = TRUE) a plain list of one plot per column.

Examples

```
xpdb_x %>%
  backfill_shk() %>%
  shk_vs_catcov()

# Combine all shk columns onto one shared, faceted plot instead of a list
xpdb_x %>%
  backfill_shk() %>%
  shk_vs_catcov(list = FALSE)
```

shk_vs_contcov *Shrinkage contribution versus continuous covariates*

Description

Mirrors `eta_vs_contcov()`, but for the per-individual shrinkage contribution diagnostic (shk type columns, see `derive_shk()`/`backfill_shk()`) instead of the etas themselves.

Usage

```
shk_vs_contcov(
  xpdb,
  mapping = NULL,
  shkvar = NULL,
  covvar = NULL,
  drop_fixed = TRUE,
  linsm = FALSE,
  type = "ps",
  list = TRUE,
  title = "Shrinkage contribution versus continuous covariates | @run",
  subtitle = "Based on @nind individuals",
  caption = "@dir",
  tag = NULL,
  log = NULL,
  guide = TRUE,
  facets,
  .problem,
  quiet,
  ...
)
```

Arguments

xpdb	<xp_xtras> or <xpose_data> object
mapping	ggplot2 style mapping
shkvar	tidyselect for shk variables
covvar	tidyselect for continuous covariate variables; NULL (default) selects every continuous covariate in the xpdb data index.
drop_fixed	As in xpose
linsm	If type contains "s" should the smooth method by lm?
type	Passed to xplot_scatter
list	<logical> Only relevant when shkvar resolves to more than one shk column. If TRUE (default, for backwards compatibility), returns a plain list of one plot per shk column. If FALSE, they are instead combined onto one shared plot – faceted by shk column, in addition to the existing per-covariate facet – automatically

	paginating (at most 9 panels per page, i.e. ncol/nrow of 3) via xpose's own facet_wrap_paginate mechanism. Printing the returned plot renders every page; pass page to print() to select a specific one.
title	Plot title
subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag
log	Log scale covariate value?
guide	Add guide line?
facets	Additional facets
.problem	Problem number
quiet	Silence output
...	Any additional aesthetics.

Value

The desired plot, or (when shkvar resolves to more than one shk column and list = TRUE) a plain list of one plot per column.

Examples

```
xpdb_x %>%
  backfill_shk() %>%
  shk_vs_contcov()

# Combine all shk columns onto one shared, faceted plot instead of a list
xpdb_x %>%
  backfill_shk() %>%
  shk_vs_contcov(list = FALSE)
```

---

summarise_xpdb *Group/ungroup and summarize variables in an xpdb*

Description

group_by_x() takes an existing table and converts it into a grouped table where operations are performed "by group". ungroup() removes grouping. summarize() reduces multiple values down to a single value.

Note: this function uses xpose.xtras::edit_xpose_data, but is otherwise the same as <xpose::group_by>.

Usage

```
group_by_x(.data, ..., .problem, .source, .where)

ungroup_x(.data, ..., .problem, .source, .where)
```

Arguments

.data	An xpose database object.
...	Name-value pairs of expressions. Use NULL to drop a variable.
.problem	The problem from which the data will be modified
.source	The source of the data in the xpdb. Can either be 'data' or an output file extension e.g. 'phi'.
.where	A vector of element names to be edited in special (e.g. .where = c('vpc_dat', 'aggr_obs') with vpc).

Value

Group data in an xpose data object

---

val2lvl *Translate values to levels*

Description

This is intended to be used as a convenience function in plotting where levels are set for some variable.

Usage

```
val2lvl(vals, lvl_tbl = NULL)
```

Arguments

vals	vector of values associated with levels in lvl_tbl
lvl_tbl	tibble of levels

Value

A vector of levels corresponding to the input vector. If lvl_tbl carries an ordered attribute set to TRUE (see set_var_levels()'s .ordered argument and lvl_inord()), the result is an ordered factor.

vismodegib — *A tibble of mock data used for fitting vismodegib models*

Description

The referenced work presents two alternative modeling approaches for muscle spasm response to vismodegib. There is a mock dataset for one person, and using the provided model a 50 participant mock dataset could be generated.

Usage

```
vismodegib
```

Format

`tibble`:
An `tibble`.

Source

Generated using sup-0009 and sup-0010 from the reference.

References

Lu, T., Yang, Y., Jin, J.Y. and Kågedal, M. (2020), Analysis of Longitudinal-Ordered Categorical Data for Muscle Spasm Adverse Event of Vismodegib: Comparison Between Different Pharmacometric Models. CPT Pharmacometrics Syst. Pharmacol., 9: 96-105. doi:10.1002/psp4.12487

vismo_dtmm — *An* `xp_xtras` *example of the discrete-time Markov model of categorical vismodegib data*

Description

The referenced work presents two alternative modeling approaches for muscle spasm response to vismodegib. This is a fit of the provided discrete-time Markov model to the 50 participant mock data.

Usage

```
vismo_dtmm
```

Format

`xp_xtras`:
An `xp_xtras` object.

Source

Derived from sup-0009 and sup-0010 from the reference.

References

Lu, T., Yang, Y., Jin, J.Y. and Kågedal, M. (2020), Analysis of Longitudinal-Ordered Categorical Data for Muscle Spasm Adverse Event of Vismodegib: Comparison Between Different Pharmacometric Models. CPT Pharmacometrics Syst. Pharmacol., 9: 96-105. doi:10.1002/psp4.12487

Examples

```
# To establish as a complete categorical DV example:
vismo_dtmm  <- vismo_dtmm   %>%
  set_var_types(.problem=1, catdv=DV, dvprobs=matches("^P\\d+$")) %>%
  set_dv_probs(.problem=1, 0~P0,1~P1,ge(2)~P23)
```

`vismo_pomod` *An* `xp_xtras` *example of the proportional odds categorical vismodegib model*

Description

The referenced work presents two alternative modeling approaches for muscle spasm response to vismodegib. This is a fit of the provided proportional odds model to the 50 participant mock data.

Usage

```
vismo_pomod
```

Format

`xp_xtras`**:**

An `xp_xtras` object.

Source

Derived from sup-0009 and sup-0010 from the reference.

References

Lu, T., Yang, Y., Jin, J.Y. and Kågedal, M. (2020), Analysis of Longitudinal-Ordered Categorical Data for Muscle Spasm Adverse Event of Vismodegib: Comparison Between Different Pharmacometric Models. CPT Pharmacometrics Syst. Pharmacol., 9: 96-105. doi:10.1002/psp4.12487

Examples

```
# To establish as a complete categorical DV example:
vismo_pomod <- vismo_pomod  %>%
  set_var_types(.problem=1, catdv=DV, dvprobs=matches("^P\\d+$")) %>%
  set_dv_probs(.problem=1, 0~P0,1~P1,ge(2)~P23)
```

---

wrap_xp_ggally *Ensure consistent style with* GGally *functions*

Description

Ensure consistent style with GGally functions

Usage

```
wrap_xp_ggally(fn, xp_theme, ...)
```

Arguments

fn	<character> name of GGally function
xp_theme	theme to use
...	<any> additional arguments to pass to GGally function

Value

ggplot2 function

---

xp4_xtra_theme *Updated version of the xpose4 theme*

Description

Updated version of the xpose4 theme

Usage

```
xp4_xtra_theme()
```

Value

An xpose theme object with xpose4 color palette

xpdb_set *An example* xpose_set *object*

Description

A set of identical xpdb objects to demo various features of xpose.xtras.

Usage

```
xpdb_set
```

Format

xpose_set:
An xpose_set object of length 4 with a single lineage.

Source

Assembled from the xpdb_ex_pk object in the xpose package.

xpdb_x *An example* xp_xtras *object*

Description

The <xpdb_ex_pk> object converted to xp_xtras. For examples.

Usage

```
xpdb_x
```

Format

xp_xtras:
An xp_xtras object with no extra data filled.

Source

Assembled from the xpdb_ex_pk object in the xpose package.

`xplot_binned` *Generic binned trend plotting function*

Description

Following the xpose design pattern, this is a generic template (analogous to `xplot_boxplot()` or `xplot_pairs()`) for rendering already-summarized/binned data as connected points and/or lines across a (typically discrete, possibly ordered) x variable. It is not tied to any particular binning or summarization logic – callers are expected to shape their data (eg via opt's `post_processing`) and build a named implementation on top of it (eg `catdv_vs_occ()`) rather than calling it directly for everyday use.

Usage

```
xplot_binned(
  xpdb,
  mapping = NULL,
  group = "variable",
  type = "pl",
  xscale = "discrete",
  yscale = "continuous",
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  tag = NULL,
  plot_name = "binned",
  gg_theme,
  xp_theme,
  opt,
  quiet,
  ...
)
```

Arguments

xpdb	<xp_xtras> or <xpose_data> object
mapping	ggplot2 style mapping. Expected to include at least x and y.
group	Column name distinguishing multiple summarized series (eg a variable column). Points/lines/smooths are both connected and coloured by this column (see `xpose::xplot_scatter()` for the same connecting-line convention).
type	String setting the type of plot to be used: point p, line l, and smooth s, or any combination thereof.
xscale	Defaults to discrete.
yscale	Defaults to continuous.
title	Plot title

subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag
plot_name	Metadata name of plot
gg_theme	As in xpose
xp_theme	As in xpose
opt	Processing options for fetched data
quiet	Silence extra debugging output
...	Additional aesthetics, passed to the point, line and smooth layers.

Details

Unlike xpose::xplot_scatter()'s raw per-subject spaghetti plots, xplot_binned() assumes the supplied data is already one row per x/group combination (eg per occasion, per series) – it does no binning, aggregation, or tidying of its own.

Value

The desired plot

---

xplot_boxplot *Default xpose boxplot function*

Description

Manually generate boxplots from an xpdb object.

Usage

```
xplot_boxplot(
  xpdb,
  mapping = NULL,
  type = "bo",
  xscale = "discrete",
  yscale = "continuous",
  orientation = "x",
  group = "ID",
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  tag = NULL,
  plot_name = "boxplot",
  gg_theme,
  xp_theme,
  opt,
```

```
  quiet,
  jitter_seed,
  ...
)
```

Arguments

xpdb	<xp_xtras> or <xpose_data> object
mapping	ggplot2 style mapping
type	See Details.
xscale	Defaults to discrete.
yscale	Defaults to continuous, used as check if orientation changed.
orientation	Defaults to x
group	Grouping for connecting lines through jitter
title	Plot title
subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag
plot_name	Metadata name of plot
gg_theme	As in xpose
xp_theme	As in xpose
opt	Processing options for fetched data
quiet	Silence extra debugging output
jitter_seed	A numeric, optional seed to be used in jitters
...	Any additional aesthetics.

Details

For type-based customization of plots:

- b box-whisker (using default quantiles)
- p points (from geom_dotplot)
- v violin (from geom_violin)
- o outliers (show outliers)
- l line through 0 (or as indicated in hline_yintercept or yline_xintercept)
- s smooth line (from geom_smooth)
- j jitter points (from geom_jitter)
- c connecting lines for jitter points (from geom_path)

Value

The desired plot

xplot_forest *Default xpose forest plot function*

Description

Manually generate a forest-style plot (point + interval per category) from an xpdb object. This is a generic, low-level renderer in the same spirit as xplot_boxplot()/xpose::xplot_scatter(): it has no built-in knowledge of covariate associations, prm_cov(), or any other specific data source – it just draws a point + interval (and, per type, a reference guide line) from whatever mapping/pre-fetched opt data it's given. See cov_forest() for the covariate-association-specific wrapper that prepares that mapping from prm_cov() and calls this function to render it.

Usage

```
xplot_forest(
  xpdb,
  mapping = NULL,
  type = "pi",
  region = NULL,
  orientation = "y",
  xscale = "continuous",
  yscale = "discrete",
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  tag = NULL,
  plot_name = "forest",
  gg_theme,
  xp_theme,
  opt,
  violin_opt,
  quiet,
  ...
)
```

Arguments

xpdb	<xp_xtras> or <xpose_data> object
mapping	ggplot2 style mapping. Expected aesthetics: x/y (the point) and xmin/xmax (the interval), or the mirrored roles when orientation = "x". For the violin layer (type includes "v"), also needs violin_x/violin_y (prefixed, since this layer's data – from violin_opt – has a different shape than the rest and can't share the plain x/y mapping; see xpose::xp_geoms()'s {name}_{aes} convention for per-layer aesthetic overrides).
type	See Details.

region	<numeric(2)> c(low, high) bounds for the shaded "no relevant effect" region (type includes "r"), eg c(0.8, 1.25) for a bioequivalence-style band. NULL (default) falls back to c(0.8, 1.25) whenever "r" is requested; has no effect otherwise.
orientation	Defaults to 'y' (categories on the y-axis, values on the x-axis – the conventional forest-plot layout).
xscale	Defaults to 'continuous'.
yscale	Defaults to 'discrete'.
title	Plot title
subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag
plot_name	Metadata name of plot
gg_theme	As in xpose
xp_theme	As in xpose
opt	Processing options for fetched data (one row per category).
violin_opt	Processing options for the violin layer's data (one row per draw), only used/required when type includes "v". Fetched separately from opt (a distinct xpose::fetch_data() call, noted via cli::cli_inform() unless quiet = TRUE) because the two layers need different data shapes.
quiet	Silence extra debugging output
...	Any additional aesthetics, or overrides for the reference line (eg vline_xintercept = 1 for a ratio-style forest plot; defaults to 0 like the rest of the package's guide lines, see xp_xtra_theme()).

Details

For type-based customization of plots:

- p point (from geom_point) – the effect estimate
- i interval (from geom_linerange) – the confidence/credible interval
- l reference line through the theme's vline_xintercept/ hline_yintercept (0 by default; a ratio-style forest plot will typically override this to 1, see cov_forest())
- v violin/density (from geom_violin), showing the distribution behind an interval (eg simulation draws) – requires violin_opt and a violin_x/violin_y mapping, see above
- r shaded reference region (from geom_rect) spanning region (default c(0.8, 1.25)), eg a bioequivalence-style "no relevant effect" band; drawn behind every other layer

Value

The desired plot

xplot_heatmap *Generic heatmap plotting function*

Description

Following the xpose design pattern, this is a generic template (analogous to `xplot_boxplot()` or `xplot_pairs()`) for rendering a numeric matrix as a themed tile heatmap with cell value labels. It is not tied to any particular data source; callers are expected to build a named implementation on top of it (eg, `cormat()` for parameter correlation/covariance matrices) rather than calling it directly for everyday use.

Usage

```
xplot_heatmap(
  xpdb,
  mat,
  digits = 3,
  limits = NULL,
  midpoint = 0,
  legend_name = "Value",
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  tag = NULL,
  plot_name = "heatmap",
  gg_theme,
  xp_theme,
  quiet,
  ...
)
```

Arguments

xpdb	<xp_xtras> or <xpose_data> object. Used only for theming; mat is plotted as-is.
mat	A numeric matrix with row and column names. NA cells are dropped (not plotted), which can be used to mask out redundant cells (eg, the lower triangle of a symmetric matrix).
digits	Number of significant digits to display in cell labels
limits	Fill scale limits, as c(low, high). Defaults to c(-1, 1) * max(abs(mat), na.rm = TRUE)
midpoint	Fill scale midpoint
legend_name	Fill scale/legend title
title	Plot title
subtitle	Plot subtitle

caption	Plot caption
tag	Plot tag
plot_name	Metadata name of plot
gg_theme	As in xpose
xp_theme	As in xpose
quiet	Silence extra debugging output
...	Additional aesthetics, passed to the heatmap tile layer and heatmaptxt label layer (see xp_xtra_theme())

Value

The desired plot

Examples

```
m <- matrix(c(1, 0.5, 0.5, 1), nrow = 2, dimnames = list(c("A", "B"), c("A", "B")))
xplot_heatmap(xpdb_x, m, quiet = TRUE)
```

---

xplot_pairs *Wrapper around ggpairs*

Description

Following the xpose design pattern to derive <ggpairs> plots.

Established xplot_ are used to generate parts of the grid.

Usage

```
xplot_pairs(
  xpdb,
  mapping = NULL,
  cont_opts = list(group = "ID", guide = FALSE, type = "ps"),
  dist_opts = list(guide = FALSE, type = "hr"),
  cat_opts = list(type = "bo", log = NULL),
 contcont_opts = list(other_fun = NULL, stars = FALSE, digits = reportable_digits(xpdb),
    title = "Pearson Corr"),
 catcont_opts = list(other_fun = NULL, stars = FALSE, digits = reportable_digits(xpdb),
    title = "Spearman rho"),
  catcat_opts = list(use_rho = TRUE),
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  tag = NULL,
  plot_name = "pairs",
  gg_theme,
```

```
  xp_theme,
  opt,
  quiet,
  progress = rlang::is_interactive() && quiet,
  switch = NULL,
  ...
)
```

Arguments

xpdb	<xp_xtras> or <xpose_data‘> object
mapping	ggplot2 style mapping
cont_opts	List of options to pass to xplot_scatter. See Details
dist_opts	List of options to pass to xplot_distribution. See Details
cat_opts	List of options to pass to xplot_boxplot. See Details
contcont_opts	List of options to pass to ggally_cors. See Details
catcont_opts	List of options to pass to ggally_statistic. See Details
catcat_opts	A list with use_rho TRUE or FALSE. If TRUE (default), then the Spearman rho is displayed, else the ggpairs default count is used.
title	Plot title
subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag
plot_name	Metadata name of plot
gg_theme	As in xpose. This does not work reliably when changed from the default.
xp_theme	As in xpose
opt	Processing options for fetched data
quiet	Silence extra debugging output
progress	Show a progress bar as the plot is generated?
switch	Passed to ggpairs
...	Ignored

Details

There is only limited control over the underlying ggpairs() call given the need to address abstractions in GGally and xpose. However, users can modify key display features. For scatter, distribution and boxplots, the type option is directly forwarded to the user. For upper elements of the matrix, users can modify features of the text displayed or supply some other function entirely (other_fun).
_opts lists are consumed with <modifyList> from the default, so there is no need to declare preferences that align with the default if updating a subset.

Value

specified pair plot

`xplot_rocplot` *Default xpose ROC plot function*

Description

Manually generate ROCs from an xpdb object.

Usage

```
xplot_rocplot(
  xpdb,
  mapping = NULL,
  type = "c",
  guide = TRUE,
  xscale = "continuous",
  yscale = "continuous",
  group = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  tag = NULL,
  plot_name = "xplot_rocplot",
  gg_theme,
  xp_theme,
  opt,
  quiet,
  thres_fixed = 0.5,
  like_col = NULL,
  obs_col = NULL,
  obs_target = NULL,
  auc_sprintf = "AUC: %.3f",
  ...
)
```

Arguments

`xpdb`	<xp_xtras> or <xpose_data> object
`mapping`	ggplot2 style mapping
`type`	See Details.
`guide`	Should the guide (e.g. unity line) be displayed.
`xscale`	Defaults to continuous.
`yscale`	Defaults to continuous.
`group`	Grouping for curves or points
`title`	Plot title
`subtitle`	Plot subtitle

caption	Plot caption
tag	Plot tag
plot_name	Metadata name of plot
gg_theme	As in xpose
xp_theme	As in xpose
opt	Processing options for fetched data
quiet	Silence extra debugging output
thres_fixed	Fixed threshold value for space
like_col	Column for likelihood/probability value
obs_col	Column for observed value
obs_target	Target observed value for likelihood
auc_sprintf	Format to apply to AUC label
...	Any additional aesthetics.

Details

For type-based customization of plots:

- c ROC curve (using geom_path)
- k Key points on ROC curve (where on curve the threshold is thres_fixed) (using geom_point)
- p ROC space points (using geom_point)
- t ROC space text (using geom_text)
- a AUC in bottom right (using geom_label)

Value

The desired plot

---

xpose_set *Generate a set of* xpdb *objects*

Description

This function generates a set of xpose data (xpdb) objects that can be used to define relationships between models. The

Usage

```
xpose_set(..., .relationships = NULL, .as_ordered = FALSE)
```

Arguments

`...`	<dynamic-dots> `xpdb1, xpdb2, ...` A set of `xpdb` objects to be combined into a set.
`.relationships`	<list> A list of relationships between the `xpdb` objects. (see Details)
`.as_ordered`	<logical> Alternative to `.relationships`, should the set of `xpdb` objects provided be considered a lineage (`grandparent, parent, child, ...`)?

Details

Beyond just a list of `xpdb` objects, an `xpose_set` adds hierarchical information.

When using `.relationships`, these should be expressed as tilde formulas, where the left-hand side is children and the right and side is parents. In the simplest case, this would be `child ~ parent`, but a child can have multiple parents. This syntax expects that the names for models is either declared as argument names in the call, or that the variable names are directly used (i.e., not spliced or passed as an unnamed list).

Value

A list of class `xpose_set`

Examples

```
data("xpdb_ex_pk", package = "xpose")

# Arbitrary copy
xpdb_ex_pk2 <- xpdb_ex_pk

# Simplest call
set1 <- xpose_set(xpdb_ex_pk, xpdb_ex_pk2)

# With predefined relationships
set2 <- xpose_set(xpdb_ex_pk, xpdb_ex_pk2,
  .relationships = list(xpdb_ex_pk2 ~ xpdb_ex_pk)
  )

# Alternative predefined relationships
set2b <- xpose_set(xpdb_ex_pk, xpdb_ex_pk2,
  .as_ordered = TRUE
  )

# With custom labels
set3 <- xpose_set(mod1 = xpdb_ex_pk, mod2 = xpdb_ex_pk2,
  .relationships = list(mod2 ~ mod1)
  )

# Alternative set3 using dyanmic dots
mod_list <- list(
  mod1 = xpdb_ex_pk,
  mod2 = xpdb_ex_pk2
)
```

```
mod_rels <- list(
  mod2 ~ mod1
)
set3b = xpose_set(!!!mod_list, .relationships = mod_rels)
```

---

xp_from_bbr *Convenience function for ingesting a bbr model into xpose and xpose.xtras*

Description

bbr is a Metrum Research Group package for managing NONMEM modeling workflows via bbi. It is not distributed on CRAN (see Additional_repositories in this package's DESCRIPTION for where to obtain it).

Reading a bbr-managed model into xpose/xpose.xtras normally requires manually reconstructing the output file path from the model object, e.g.

```
xtras_data(
  file = file.path(bbr::get_output_dir(mod), paste0(bbr::get_model_id(mod), ".lst"))
)
```

xp_from_bbr() wraps that pipeline.

Usage

```
xp_from_bbr(.mod, ..., .use_bbr_descr = TRUE)
```

Arguments

.mod	<bbi_nonmem_model> A bbr NONMEM model object, e.g. as returned by bbr::read_model().
...	Passed to xtras_data() (and, in turn, xpose::xpose_data()).
.use_bbr_descr	<logical> If TRUE (default) and .mod carries a bbr description, use it to set the descr property of the result (see set_prop()), taking precedence over any description parsed from the NONMEM output itself.

Value

An <xp_xtras> object

See Also

bbr::read_model(), xtras_data()

Examples

```
if (requireNamespace("bbr", quietly = TRUE)) {
  # Build a bbr-style model directory from the bundled pheno_saemimp example
  src_dir <- system.file("pheno_saemimp", package = "xpose.xtras")
  mod_dir <- tempfile("xp_from_bbr_ex")
  dir.create(mod_dir)
  file.copy(file.path(src_dir, "run18.mod"), file.path(mod_dir, "18.mod"))
  out_dir <- file.path(mod_dir, "18")
  dir.create(out_dir)
  out_files <- setdiff(list.files(src_dir, pattern = "^run18\\."), "run18.mod")
  for (f in out_files) {
    file.copy(
      file.path(src_dir, f),
      file.path(out_dir, paste0("18.", sub("^run18\\.", "", f)))
    )
  }
  # bbr considers a run finished once bbi has written this file
  writeLines("{}", file.path(out_dir, "bbi_config.json"))

 mod <- bbr::new_model(file.path(mod_dir, "18"), .description = "Phenobarbital SAEM model")
  print(xp_from_bbr(mod))
  unlink(mod_dir, recursive = TRUE)
}
```

---

xp_var xp_var *Method*

Description

To add a small amount of functionality to <xp_var>, this method was created.

Usage

```
xp_var(xpdb, .problem, col = NULL, type = NULL, silent = FALSE)

## Default S3 method:
xp_var(xpdb, .problem, col = NULL, type = NULL, silent = FALSE)

## S3 method for class 'xp_xtras'
xp_var(xpdb, .problem, col = NULL, type = NULL, silent = FALSE)
```

Arguments

xpdb	An xpose database object.
.problem	The $problem number to be used.
col	The column name to be searched in the index. Alternative to arg ‘type‘.
type	The type of column to searched in the index. Alternative to ‘col‘.
silent	Should the function be silent or return errors.

Value

A tibble of identified variables.

---

xp_xtra_theme *Extra theme defaults*

Description

Adds aesthetics for plot components used in this package.

Usage

```
xp_xtra_theme(base_on = NULL)
```

Arguments

base_on	xp_theme object to extend

Details

This package attempts to generate a consistent theme even if users are working with a highly customized xp_theme. There is are only a few hard-coded aesthetics, and the rest are derived from existing aesthetics in base_on, which defaults to the default from xpose.

Only a few options are worth noting. In <xplot_pairs> (and functions using it), the aesthetics for GGally-specific elements like barDiag are defined as gga(element)_(aesthetic). The labeller for pairs plots is also changed from the *de facto* default label_both to label_value, but any labeller can be provided as pairs_labeller.

Value

An xpose theme object

---

xset_lineage *Determine lineage within a set*

Description

Determine lineage within a set

Usage

```
xset_lineage(xpdb_s, ..., .spinner = NULL)
```

Arguments

xpdb_s	<xpose_set> object
...	<dynamic-dots> labels for models in the set from which to create lineages (will result in a list if multiple labels are used). If empty, lineage from base model will be output; if no base, first listed model will be used. Always used the most senior model in this list.
.spinner	Set to FALSE to not show a loading spinner in interactive mode.

Details

This function uses a not-especially-optimized tree-searching algorithm to determine the longest lineage starting from whatever is treated as the base model. It is based loosely on <pluck_depth>, but the values at each depth are maintained. As such, for larger sets this function and, more importantly, functions that use it may take some time.

Value

<character> vector of c('base', 'base child', 'base grandchild', ...) or list thereof, depending on dots arguments.

Examples

```
xset_lineage(xpdb_set)

set_base_model(xpdb_set, fix1) %>%
  xset_lineage()

xset_lineage(xpdb_set, fix1)
```

---

xset_waterfall *Waterfall plot*

Description

Generic function primarily used with wrappers targeting types of values changed between two models.

Usage

```
xset_waterfall(
  xpdb_s,
  ...,
  .inorder = FALSE,
  type = "bh",
```

```
  .cols = NULL,
  max_nind = 0.7,
  scale_diff = TRUE,
  show_n = TRUE,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  tag = NULL,
  plot_name = "waterfall",
  opt,
  facets = NULL,
  facet_scales = "free_x",
  .problem,
  .subprob,
  .method,
  quiet
)
```

Arguments

xpdb_s	<xpose_set> object
...	See <two_set_dots>
.inorder	See <two_set_dots>
type	See Details.
.cols	<tidyselect> data columns to plot.
max_nind	If less than 1, the percentile of absolute change values above which to plot. If above 1, the absolute number of subjects is included. To show all, use an extreme positive number like 9999.
scale_diff	<logical> Scale change to the standard deviation of the model 1 column values. Respects faceting.
show_n	<logical> For faceting variables, show N per facet. *Not implemented*
title	Plot title
subtitle	Plot subtitle
caption	Plot caption
tag	Plot tag
plot_name	Metadata name of plot
opt	User-specified data options. Only some of these will be used.
facets	<character> Faceting variables
facet_scales	<character> Forwarded to facet_*(scales = facet_scales)
.problem	The problem to be used, by default returns the last one.
.subprob	The subproblem to be used, by default returns the last one.
.method	The estimation method to be used, by default returns the last one.
quiet	Silence extra debugging output

Details

For type-based customization of plots:

- b bar plot (from geom_bar)
- h hline at 0 (from geom_hline)
- t text of change value (from geom_text)

Value

The desired plot

---

xtras_data *Read model outputs directly into an* xp_xtras *object*

Description

Convenience wrapper equivalent to xpose::xpose_data(...) %>% as_xp_xtras(). xpose::xpose_data()'s quiet argument already controls its own informative messages, but not warnings raised while parsing NONMEM tables – e.g. readr surfacing every oddly formatted or NaN/Inf value it had to coerce, one warning per table, which can drown out a warning that actually matters. dplyr (>= 1.1.2, already required by this package) batches any warnings raised inside a single dplyr::mutate() call – which is how xpose reads each table – into one rlang_warning per call rather than letting each one through individually.

By default, xtras_data() silences those (and only those) warnings; pass warn = TRUE to see them as xpose::xpose_data() would raise them. Warnings xpose::xpose_data() raises directly – e.g. a table or output file it couldn't find at all – are unaffected either way, since those indicate an actual problem rather than value-level noise.

Usage

```
xtras_data(..., warn = FALSE)
```

Arguments

...	Passed to xpose::xpose_data()
warn	<logical> If FALSE (default), rlang_warning-class warnings raised while reading are silenced. If TRUE, all warnings are passed through unmodified.

Value

An <xp_xtras> object

See Also

xpose::xpose_data(), as_xpdb_x()

Examples

```
xtras_data(file = file.path(
  system.file("pheno_saemimp", package = "xpose.xtras"), "run18.lst"
))
```

%p% *Binary check if LHS is parent of RHS*

Description

Binary check if LHS is parent of RHS

Usage

```
possible_parent %p% possible_child
```

Arguments

possible_parent	<xpose_set_item> object suspected as parent to ...
possible_child	... <xpose_set_item> object suspected child

Value

<logical> TRUE if LHS is parent of RHS

Examples

```
# Detect direct parent
pheno_set$run6 %p% pheno_set$run7

# Detect non-parentage (does not try to "flip" parentage)
pheno_set$run6 %p% pheno_set$run5

# Does not detect grand-parentage
pheno_set$run6 %p% pheno_set$run13
```

Index