Package 'xpose.xtras'

April 22, 2026

Title Extra Functionality for the 'xpose' Package

Version 0.1.4

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

RoxygenNote 7.3.3

Imports checkmate, cli, colorspace, conflicted, dplyr (>= 1.1.2), forcats (>= 1.0.0), GGally, ggplot2 (>= 3.4.2), glue, 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 DiagrammeR, grDevices, knitr, nlmixr2, nlmixr2data, nlmixr2est, 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

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 2026-04-21 22:30:02 UTC

Contents

`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_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_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)
```

as_leveler *Level-defining helper functions*

Description

Level-defining helper functions

Usage

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

is_leveler(x)

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

lvl_sex()

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

Arguments

x	<character> vector of levels
.start_index	<numeric> starting index for levels

Value

Special character vector suitable to be used as leveler

Examples

```
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)
```

`check_levels` *Verify validity of level list*

Description

Verify validity of level list

Usage

```
check_levels(lvl_list, index)
```

Arguments

`lvl_list`	`<list>` of formulas or leveler functions
`index`	Index of `xp_xtras` object

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)
```

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)
```

`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,
  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
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,
  drop_fixed = TRUE,
  orientation = "x",
  show_n = check_xpdb_x(xpdb, .warn = FALSE),
  type = "bol",
  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
`drop_fixed`	As in xpose
`orientation`	Passed to xplot_boxplot
`show_n`	Add "N=" to plot
`type`	Passed to xplot_boxplot
`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

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()
```

`eta_vs_contcov` *Eta continuous covariate plots (typical)*

Description

Eta continuous covariate plots (typical)

Usage

```
eta_vs_contcov(
  xpdb,
  mapping = NULL,
  etavar = NULL,
  drop_fixed = TRUE,
  linsm = FALSE,
  type = "ps",
  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
drop_fixed	As in xpose
linsm	If type contains "s" should the smooth method by lm?
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?
facets	Additional facets
.problem	Problem number
quiet	Silence output
...	Any additional aesthetics.

Value

The desired plot

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()
```

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.

Value

An xpose_set object with the focused xpdb object(s)

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()

# 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()
```

---

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")
```

`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_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

For xpose version > 0.5.0 **[Deprecated]**

Because this has been fixed in the parent package, the fix will be removed in an upcoming release.

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.

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)
```

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 the all problems if NULL
.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)
```

---

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"`.

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()`.

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)
```

---

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

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)
```

`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_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 the all problems if `NULL`
`...`	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()
```

---

set_option *Set an* xpose *option*

Description

Set an xpose option

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",
  .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
`.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)
  )
```

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")
```

---

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.

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_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_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_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

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

Description

Binary check if LHS is parent of LHS

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