Package 'depigner'

January 11, 2021

Title A Utility Package to Help you Deal with ``Pignas''

Version 0.8.4

Description Pigna [_pìn'n'a_] is the Italian word for pine cone. In jargon, it is used to identify a task which is boring, banal, annoying, painful, frustrating and maybe even with a not so beautiful or rewarding result, just like the obstinate act of trying to challenge yourself in extracting pine nuts from a pine cone, provided that, in the end, you will find at least one inside it. Here you can find a backpack of functions to be used to solve small everyday problems of coding or analyzing (clinical) data, which would be normally solved using quick-and-dirty patches. You will be able to convert 'Hmisc' and 'rms' summary()es into data.frames ready to be rendered by 'pander' and 'knitr'. You can access easy-to-use wrappers to activate essential but useful progress bars (from 'progress') into your loops or functionals. Easy setup and control Telegram's bots (from 'telegram.bot') to send messages or to divert error messages to a Telegram's chat. You also have some utilities helping you in the development of packages, like the activation of the same user interface of 'usethis' into your package, or call polite functions to ask a user to install other packages. Finally, you find a set of thematic sets of packages you may use to set up new environments quickly, installing them in a single call.

License GPL-3

URL https://corradolanera.github.io/depigner/, https://github.com/CorradoLanera/depigner

BugReports https://github.com/CorradoLanera/depigner/issues

Depends R (>= 3.6)

Imports desc, dplyr (>= 0.7.7), fs, ggplot2 (>= 3.1.0), Hmisc (>= 4.2.0), magrittr (>= 1.5), progress (>= 1.2.0), purrr (>= 0.2.5), rlang (>= 0.2.2), rms (>= 5.1.2), rprojroot, stats, stringr (>= 1.3.1), telegram.bot (>= 2.3.0), tibble (>= 1.4.2), tidyr (>= 0.8.1), usethis (>= 1.5.0), utils

Suggests covr (>= 3.1.0), spelling, survival, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.1.1

NeedsCompilation no

Author Corrado Lanera [aut, cre, cph]

Maintainer Corrado Lanera <corrado.lanera@gmail.com>

Repository CRAN

Date/Publication 2021-01-11 06:50:12 UTC

R topics documented:

adjust_p *Adjust P-values*

Description

Adjust P-values of a tidy_summary object

Usage

```
adjust_p(x, method)

## S3 method for class 'tidy_summary'
adjust_p(x, method = "BH")
```

Arguments

x	a tidy_summary object.
method	(chr, default = "BH") a valid method for p.adjust

Value

a tidy_summary object with the Ps adjusted

Examples

```
library(Hmisc)
my_summary <- summary(Species ~ ., data = iris,
  method = "reverse",
  test = TRUE
)

tidy_summary(my_summary, prtest = "P") %>%
  adjust_p()
```

Arthritis *Data for example*

Description

This data are imported from the package vcd, which depends on a package (i.e., mvtnorm) which require R >= 3.5. For this reason we do not import the package vcd, but only this dataset. Which is used only in the examples and tests.

Usage

```
Arthritis
```

Format

A data.frame with 84 obs. and 5 var.

`check_for_bot_options` *Check if a bot is set up*

Description

`check_for_bot_options` check if a telegram bot and corresponding chat, exist.

Usage

```
check_for_bot_options(chat_id, bot)
```

Arguments

`chat_id`	a chat id
`bot`	a bot

Value

invisible lgl indicating if check pass (TRUE) or not (FALSE)

`ci2p` *ci2p*

Description

`ci2p` compute the p-value related with a provided confidence interval. It considers a symmetric distribution (by default standard normal).

Usage

```
ci2p(
  est,
  lower,
  upper,
  log_transform = FALSE,
  conf = 0.95,
  qdist = stats::qnorm,
  pdist = stats::pnorm
)
```

Arguments

est	estimated value
lower	lower bound of the confidence level
upper	upper bound of the confidence level
log_transform	(default 'FALSE') flag indicating if a log transformation as to apply to the data
conf	(default '95%') confidence level
qdist	(default 'qnorm') quantile function
pdist	(default 'pnorm') distribution function

Details

Confidence level can be customize (by default 95 possible to apply a log transformation in case of proportions.

Value

a p-value

Examples

```
ci2p(1.125, 0.634, 1.999, log_transform = TRUE)
ci2p(1.257, 1.126, 1.403, log_transform = TRUE)
```

---

errors_to_telegram *Divert output errors to the telegram bot*

Description

Divert output errors to the telegram bot

Usage

```
errors_to_telegram(
  chat_name = Sys.getenv("R_telegram_error_chat_name"),
  bot_name = getOption("depigner.bot_name")
)
```

Arguments

chat_name	a
bot_name	a

Value

a

Examples

```
if (interactive()) {
  library(depigner)
  errors_to_telegram()
}
```

---

gdp *GDP*

Description

A wrapper to relax

Usage

```
gdp(time = 4L, freq = 0.05)
```

Arguments

time	[num] number of seconds needed to relax yourself (default = 4)
freq	[num] rotating frequency (default = 0.05/s)

Value

invisible()

Examples

```
if (interactive()) {
  gdp()

  gdp(5)

  required_seconds_of_relax <- 10 # AKA: "conta fino a dieci!"
  rate_of_anxiety <- 1/1000       # AKA: "mi girano a mille!"
  gdp(required_seconds_of_relax, rate_of_anxiety)
}
```

htype *Type's checks accordingly to [Hmisc] package*

Description

These functions decide and report if a single variable represented by a single instance of an 'Hmisc''s [describe][Hmisc::describe] object will considered a categorical variable or a continuous one.

Usage

```
htype(x, n_unique = 10L)

htypes(x, n_unique = 10L)

## S3 method for class 'describe'
htypes(x, n_unique = 10L)

## Default S3 method:
htypes(x, n_unique = 10L)

is_hcat(x)

is_hcon(x, n_unique = 10L)
```

Arguments

x	an instance of the class [describe][Hmisc::describe], in the cases of "singular" functions (ie 'is_*()' or 'htype()') it must be a single-variable [describe][Hmisc::describe] object.
n_unique	(int, 10L) the minimum number of distinct values a numeric variable must have before plot.describe uses it in a continuous variable plot.

Details

A "single" object of 'Hmisc''s [describe][Hmisc::describe] class represents a variable. When you plot and object of class [describe][Hmisc::describe] the plot function decide if it is a continuous variable or a categorical one to plot it in the correspond plot. It is also possible that the variable is not considered in none of that categories, in which case it will not be plotted at all.

These functions have been produced/deduced from reading the source code of 'Hmisc''s [plot][Hmisc::describe]. In particular, from the definition of the (two distinct) functions 'f' defined within it (one for categorical variables and the other for continuous variables). Both lead to a possible execution of 'warning("no categorical variable found")' or 'warning("no continuous variable found")'. I tried to keep the same names/code/logic that I found there.

Value

(chr) ‘htype‘ returns one of "cat" (if ‘x‘ will be considered categorical), "con" (if ‘x‘ will be considered continuous), "none" (if ‘x‘ will be not considered categorical nor continuous, and hence it will be not plotted), or "both" (with a warning, if the variable will be considered both categorical and continuous. This would possibly never happen).

(chr) character vector of the types identified by [htype] for every variable represented in (each element of) ‘x‘.

(lgl) ‘is_hcat‘ returns TRUE if x will be considered categorical.

(lgl) ‘is_hcon‘ returns TRUE if x will be considered continuous.

Functions

- `htypes`: Report types for multi-variables objects
- `is_hcat`: Check if a single-instance of a [describe][Hmisc::describe] object is categorical.
- `is_hcon`: Check if a single-instance of a [describe][Hmisc::describe] object is continuous.

See Also

[describe][Hmisc::describe],

[is_hdesc], [is_single_hdesc]

Gist with test and usage examples: https://bit.ly/htype-gist

Examples

```
library(Hmisc)
desc <- describe(mtcars)

htype(desc[["vs"]]) # "cat"
htype(desc[["mpg"]]) # "con"
htype(desc[["cyl"]]) # "none"


htypes(desc) # c(
#   mpg = "con", cyl = "none", disp = "con",
#   hp = "con", drat = "con", wt = "con", qsec = "con",
#   vs = "cat", am = "cat", gear = "none",
#   carb = "none"
# )


htypes(mtcars) # htypes(desc)
htypes(letters) # "none"


is_hcat(desc[["vs"]]) # TRUE
is_hcat(desc[["mpg"]]) # FALSE
```

```
is_hcon(desc[["vs"]]) # FALSE
is_hcon(desc[["mpg"]]) # TRUE
```

`imported_from` *Packages imported by a package*

Description

Packages imported by a package

Usage

```
imported_from(x, include_self = FALSE)
```

Arguments

`x`	(chr) the package you would like to see its imports
`include_self`	(lgl, FALSE) do you like to include ‘x‘ itself in the output?

Value

(chr) vector of packages imported by ‘x‘

Examples

```
imported_from("depigner")
```

`install_pkg_set` *Check basic installed packages*

Description

This function is used to politely ask the user to install requiring packages. It is intended to be used in interactive sessions only.

Usage

```
install_pkg_set(set = pkg_all, dependencies = TRUE)
```

Arguments

`set`	(chr) packages’ names
`dependencies`	do you want to install the dependencies?

Details

You can pass arbitrarily sets of packages; on the other hands, you can use some sets already prepared and included into ‘depigner‘ (see ‘?pkg_sets‘).

Value

invisible character vector of the subset of ‘interested‘ which was not already present, and installed.

Note

By default this function install all the packages listed in ‘pkg_all‘, which is the union of all the sets of packages listed in ‘?pkg_sets‘.

See Also

please_install, pkg_sets

Examples

```
## Not run:
  install_pkg_set() # to install all the `?pkg_all`

## End(Not run)
```

---

is_hdesc *Checks for describe objects*

Description

These two function are useful to test if an object is of class [Hmisc][Hmisc::describe].

Usage

```
is_hdesc(x)

is_single_hdesc(x)
```

Arguments

x	an object to test if it is of class ‘describe‘.

Details

In ‘Hmisc‘ both "single" ‘describe‘ objects and lists of them are of class ‘describe‘. In particular, even if ‘Hmisc::describe()‘ results in a single variable description, it is directly the "single" ‘describe‘ object and not a list of them with only a single ‘describe‘ object included!

‘is_hdesc()‘ test for general inheritance.

‘is_single_hdesc()‘ test for single instance of a ‘describe‘ object.

Value

(lgl) is ‘x‘ (a single element or a general) ‘describe‘ object?

See Also

[describe][Hmisc::describe]

[is_hcat], [is_hcon], [htype], [htypes]

Examples

```
library(Hmisc)
desc <- describe(mtcars)

is_hdesc(desc) # TRUE
is_hdesc(desc[[1L]]) # TRUE


is_single_hdesc(desc) # FALSE
is_single_hdesc(desc[[1L]]) # TRUE
```

---

paired_test_categorical

Paired test for categorical variables

Description

Statistical tests for paired categorical variable.

Usage

```
paired_test_categorical(tab)
```

Arguments

tab	a frequency table (an integer ‘table‘, if a ‘matrix‘ is provided, it will be coerced to a ‘table‘ internally)

Details

If the test is requested for two paired groups, the `mcnemar.test` is used.

If the test is requested for more than two paired groups, the test based on Cochran-Mantel-Haenzen for repeated measures is used (powered by `mantelhaen.test`)

Value

A list with components ‘P‘ (the computed P-value), ‘stat‘ (the test statistic, either t or F), ‘df‘ (degrees of freedom), ‘testname‘ (test name), ‘statname‘ (statistic name), ‘namefun‘ ("paired_tstat", "rep_aov"), ‘latexstat‘ (LaTeX representation of statname), ‘plotmathstat‘ (for R - the plotmath representation of ‘statname‘, as a character string), ‘note‘ (contains a character string note about the test).

Note

This function could be used as ‘catTest‘ option in the `summary.formula` with method ‘reverse‘.

Examples

```
library(Hmisc)

data(Arthritis)

## two groups
summary(Treatment ~ Sex,
  data = Arthritis,
  method = "reverse",
  test = TRUE,
  catTest = paired_test_categorical
)

## more than two groups
summary(Improved ~ Sex,
  data = Arthritis,
  method = "reverse",
  test = TRUE,
  catTest = paired_test_categorical
)
```

---

paired_test_continuous

Paired test for continuous variables

Description

Statistical tests for paired continuous variable.

Usage

```
paired_test_continuous(group, x)
```

Arguments

group	(fct) vector of groups
x	(num) vector of observations. Note: length of 'x' is considered equal to the number of subjects by the number of groups. Observation must be provided by subject (e.g. c(a1, b1, c1, a2, b2, c2, a3, b3, c3, a4, b4, c4), where the letters, a, b, c, and d represents the groups and the numbers represents the patients' ids). Note only patient with observation in all the levels considered will be used.

Details

If the test is requested for two paired groups, the `t.test` is used.

If the test is requested for more than two groups, the test based on ANOVA for repeated measures is used (powered by `aov`)

Value

A list with components 'P' (the computed P-value), 'stat' (the test statistic, either t or F), 'df' (degrees of freedom), 'testname' (test name), 'statname' (statistic name), 'namefun' ("paired_tstat", "rep_aov"), 'latexstat' (LaTeX representation of statname), 'plotmathstat' (for R - the plotmath representation of 'statname', as a character string), 'note' (contains a character string note about the test).

Note

This function could be used as 'conTest' option in the `summary.formula` with method 'reverse'.

Examples

```
library(Hmisc)

## two groups
summary(Species ~ .,
  data = iris[iris[["Species"]] != "setosa", ],
  method = "reverse",
  test = TRUE,
  conTest = paired_test_continuous
)

## more than two groups
summary(Species ~ .,
  data = iris,
  method = "reverse",
  test = TRUE,
  conTest = paired_test_continuous
)

## without Hmisc
two_obs <- iris[["Sepal.Length"]][iris[["Species"]] != "setosa"]
two_groups <- iris[["Species"]][iris[["Species"]] != "setosa"]
```

```
paired_test_continuous(two_groups, two_obs)

obs <- iris[["Sepal.Length"]]
many_groups <- iris[["Species"]]
paired_test_continuous(many_groups, obs)
```

---

pb_len *Progress bar of given length*

Description

Simple wrapper for progress_bar for standard and quickly ready progress bars; including messages, bar progression, percentage and time elapsed, and ETA.

Usage

```
pb_len(.x, width = 76L, show_after = 2L, clear = FALSE)

tick(pb, what = "")
```

Arguments

.x	(int) total number of step to count
width	(int, default = 76) total console width used by the bar
show_after	(num, default = 2) minimum number of seconds needed for the process to display the progress bar
clear	(lgl, default = FALSE) if TRUE, at the end of the process the progress bar will be cleared
pb	an object of class progress_bar
what	(chr, default = "") short prompt to see at the beginning of the progressbar

Value

a progress_bar object

Functions

- tick: wrapper function to update the progress bar

Examples

```
pb <- pb_len(100)
for (i in 1:100) {
  Sys.sleep(0.1)
  tick(pb, paste("i = ", i))
}
```

pkg_sets *Packages' sets*

Description

These are presets of packages which can be used to easily prepare systems installing them in groups or all together (using 'pkg_all').

Usage

```
pkg_misc

pkg_utils

pkg_speed

pkg_mlt

pkg_stat

pkg_stan

pkg_devel

pkg_docs

pkg_prod

pkg_all
```

Format

An object of class `character` of length 3.

An object of class `character` of length 4.

An object of class `character` of length 4.

An object of class `character` of length 6.

An object of class `character` of length 7.

An object of class `character` of length 2.

An object of class `character` of length 12.

An object of class `character` of length 4.

An object of class `character` of length 4.

An object of class `character` of length 44.

Details

Packages imported by the mentioned packages are not included into the sets (that is the reason why, e.g., you do not see ‘shiny‘ or ‘knitr‘, ...).

Functions

- pkg_misc: Packages for side-interest purposes, including ‘beepr‘, ‘fortunes‘, and ‘telegram.bot‘.
- pkg_utils: Utilities for data analyses, including ‘here‘, ‘lobstr‘, ‘progress‘, and ‘tidyverse‘.
- pkg_speed: Are you in a hurry?! This includes ‘furrr‘, ‘parallel‘, ‘snow‘, and ‘tidyfast‘.
- pkg_mlt: Machine Learning in R (we do not include ‘keras‘ nor ‘tensorflow‘ which require _ad hoc_ configuration). This includes ‘caret‘, ‘glmnet‘, ‘mlr3‘, ‘snow‘, ‘tidymodels‘, ‘tidytext‘, and ‘tm‘.
- pkg_stat: (Clinical oriented) statistical analyses in R. This includes ‘CBPS‘, ‘cobalt‘, ‘MatchIt‘, ‘mice‘, ‘rms‘, ‘twang‘, and ‘WeightIt‘.
- pkg_stan: Stan analyses in R (Alert: these packages require ‘stan‘ be installed into the system. They are not included into ‘pkg_all‘). This includes ‘rstanarm‘, and ‘brms‘.
- pkg_devel: Develop stuff in R. This includes ‘assertive‘, ‘covr‘, ‘lintr‘, ‘fs‘, ‘profvis‘, ‘pryr‘, ‘renv‘, ‘roxygen2‘, ‘styler‘, ‘testthat‘, ‘spelling‘, and ‘usethis‘.
- pkg_docs: Write and render things in R. This includes ‘blogdown‘, ‘DT‘, ‘pander‘, and ‘pkgdown‘.
- pkg_prod: for develop and put in production stuffs (‘shiny‘ app included). This includes ‘golem‘, ‘docopt‘, ‘plumbr‘, and ‘shinyjs‘.
- pkg_all: Union of the ‘pkg_misc‘, ‘pkg_utils‘, ‘pkg_speed‘, ‘pkg_mlt‘, ‘pkg_stat‘, ‘pkg_devel‘, ‘pkg_docs‘, and ‘pkg_prod‘ pkg_* sets.

See Also

install_pkg_set

---

please_install *Please install*

Description

A polite helper for installing and update packages (quite exactly) inspired from a function used by Hadley Wickham at ‘RStudio::conf 2018 - San Diego‘.

Usage

```
please_install(pkgs, install_fun = install.packages, ...)
```

Arguments

pkgs	character vector of package(s) to install
install_fun	function to use for installing package(s)
...	further options for install_fun

Value

invisible

`send_to_telegram` *Send something to telegram*

Description

This is a generic function to send some object to telegram.

Usage

```
send_to_telegram(x, ..., chat_id, bot)

## S3 method for class 'character'
send_to_telegram(
  x,
  type = c("message", "photo", "document", "audio", "animation", "video", "voice",
    "sticker", "location", "videonote"),
  ...,
  chat_id = getOption("depigner.chat_id"),
  bot = getOption("depigner.bot")
)

## S3 method for class 'gg'
send_to_telegram(
  x,
  fileext = c("png", "pdf", "jpeg", "tiff", "bmp"),
  ...,
  chat_id = getOption("depigner.chat_id"),
  bot = getOption("depigner.bot")
)
```

Arguments

`x`	object to send (often a character string)
`...`	further argument to pass to the sending methods (see `Bot` for specifications.)
`chat_id`	chat_id in which send the object. By default all the methods consider the chat_id defined in the previous run of `start_bot_for_chat`.
`bot`	(`Bot`) the bot object. By default all the methods consider the bot created by a previous run of `start_bot_for_chat`.
`type`	(chr, default = "message") the type of text represent `x`. I.e., if not "message", it is normally considered as a path to the corresponding object. For further clarification see `Bot` help page.
`fileext`	(chr, default "png") one of the possible file format supported: "png", "pdf", "jpeg", "tiff" or "bmp"

Details

By default it use the bot and chat_id configured by start_bot_for_chat. The user can pass a custom bot or chat_id providing them to the corresponding argument.

Value

invisible the object x.

Examples

```
## Not run:
  library(depigner)
  library(ggplot2)

  start_bot_for_chat()
  send_to_telegram("hello world")

  gg <- ggplot(mtcars, aes(x = mpg, y = hp, colour = cyl)) +
  geom_point()

  send_to_telegram(
    "following an `mtcars` coloured plot",
    parse_mode = "Markdown"
  )
  send_to_telegram(gg)

## End(Not run)
```

---

start_bot_for_chat *Set up a Telegram bot*

Description

This function set up what is necessary to telegram.bot package to be used in a more easy way by the function provided by the depigner.

Usage

```
start_bot_for_chat(
  chat_name = Sys.getenv("R_telegram_default_chat_name"),
  bot_name = getOption("depigner.bot_name")
)
```

Arguments

chat_name	(chr, NA) The name of the chat you want to the bot send.is linked to. If NA (default) it uses the *default* chat of the bot.

bot_name (chr, NULL) This argument should be left NULL. If NULL, the function bring the bot name from the environmental variable "R_telegram_bot_name". You can pass another bot's name here as a character string too (note that in this case in the .Renviron you must have an entry like 'R_TELEGRAM_BOT_<yourbotname>=.....' containing the token related to the bot).

Details

Before you can use the depigner facilities (or the telegram.bot ones) to use your bot to chat with Telegram from R, you have to set the bot up.

To set up a bot in telegram, find @BotFather on telegram. Send the message \start to it, and then send the message \newbot to it too. Next you have to follow the very simple instruction it gives you. At the end of the process, save your bot token and never share it publicly!!

After your bot is created, go to your bot default chat profile and send the message \start.

Now you can return to R and put both the bot's name and token into the .Renviron file. To access to it you can use edit_r_environ which will open the '.Renviron' file, ready to be modified.

You need to insert two lines, namely the one for your bot's name:

'R_telegram_bot_name=<name_of_my_bot>'

and one for its token:

'R_TELEGRAM_BOT_<name_of_my_bot>="1234567879:AbcD..."

Next, restart R and you are ready to use al the (simple) functionality of the depigner package, or the flexible and complete ones from the telegram.bot package.

Value

invisible()

Examples

```
## Not run:
  library(depigner)
  start_bot_for_chat()

## End(Not run)
```

---

summary_interact *summary_interact*

Description

summary_interact

Usage

```
summary_interact(
  model,
  ref,
  discrete,
  ref_min = NULL,
  ref_max = NULL,
  level = NULL,
  ...,
  digits = 3L,
  p = FALSE
)
```

Arguments

model	A model from lrm
ref	A continuous variable for which we are interested in the estimation of the OR for the various level of interaction with a discrete variable interacting with it
discrete	The discrete interacting variable
ref_min	Denominator continuous level for the Odds Ratio (i.e., the reference level), if NULL (the default)
ref_max	Numerator continuous level for the Odds Ratio (i.e., the target level)
level	A character vector of levels to show. Default (NULL) means to show all the possible levels for the discrete variable
...	for possible future development
digits	number of significant digits to print. Default is 3 Note: the datadist has to be defined for the data used in the model
p	do you want also the P-value (default = FALSE)

Value

A data frame

Examples

```
library(rms)
options(datadist = "dd")

data("transplant")

transplant <- transplant[transplant[["event"]] != "censored", ] %>%
 droplevels()
dd <- datadist(transplant)

lrm_mod <- lrm(event ~ rcs(age, 3) * (sex + abo) + rcs(year, 3),
  data = transplant
```

```
)

lrm_mod
summary(lrm_mod)
summary_interact(lrm_mod, age, sex)
summary_interact(lrm_mod, age, sex, ref_min = 60, ref_max = 80)
summary_interact(lrm_mod, age, sex,
  ref_min = 60, ref_max = 80, digits = 5L
)

summary_interact(lrm_mod, age, abo)
summary_interact(lrm_mod, age, abo, level = c("A", "AB"))
summary_interact(lrm_mod, age, abo, level = c("A", "AB"), p = TRUE)
```

`tidy_summary` *tidy_summary*

Description

Converts a `summary()` object produced by `Hmisc` or by `rms` packages to a tidy data frame ready to be ‘pander‘ed (e.g. printed on a word document after `knit`ting the source (with ‘knitr‘).

Usage

```
tidy_summary(x, ..., digits = 3L)

## S3 method for class 'summary.formula.reverse'
tidy_summary(x, ..., digits = 3L)

## S3 method for class 'summary.rms'
tidy_summary(x, ..., digits = 3L)
```

Arguments

`x`	an object used to select a method, output of some summary by `Hmisc`.
`...`	further arguments passed to or from other methods
`digits`	number of significant digits to use (default 3L).

Value

a [tibble][tibble::tibble-package]

Methods (by class)

- `summary.formula.reverse`: Tidies a summary reverse output from the `summary.formula` called with `method = "reverse"`.
- `summary.rms`: Convert the output of the `summary.rms` into a data frame, reporting only the Hazard Ratio with the .95 CI and the incremental step (for continuous variables) reference (for categorical variables) for which the Hazard is referred to (i.e. without βs, Low, High, S.E. and Type).

Note

to see the options you can pass to `...` for a custom print, see the print section in `summary.formula`.

Examples

```
library(Hmisc)
my_summary <- summary(Species ~ ., data = iris, method = "reverse")
tidy_summary(my_summary)


library(rms)
options(datadist = "dd")
n <- 1000L
set.seed(731L)
age <- 50L + 12L * rnorm(n)
sex <- factor(sample(c("Male", "Female"), n,
  rep = TRUE,
  prob = c(.6, .4)
))
cens <- 15L * runif(n)
h <- .02 * exp(.04 * (age - 50L) + .8 * (sex == "Female"))
dt <- -log(runif(n)) / h
e <- ifelse(dt <= cens, 1L, 0L)
dt <- pmin(dt, cens)

dd <- datadist(age, sex)

S <- survival::Surv(dt, e)
f <- rms::cph(S ~ age + sex)


my_summary <- summary(f)
tidy_summary(my_summary)
```

---

use_ui *Use usethis'ui(s) in your package*

Description

Does setup necessary to use the usethis' user interfaces in your package. This function requires the use roxygen. * Check that the active package uses roxygen2 * Adds usethis package to "Imports" in 'DESCRIPTION' * Imports in your namespace: - block styles: `ui_line`, `ui_todo` `ui_done`, `ui_todo`, `ui_oops` `ui_info`, `ui_code_block` - conditions: `ui_stop`, `ui_warn` - questions: `ui_yeah`, `ui_nope` - inline styles: `ui_field`, `ui_value` `ui_path`, `ui_code`) user interfaces

Usage

```
use_ui()
```

Details

Attribution: most of the source content of this function is taken and/or adapted from the corresponding unexported function in the 'usethis' package.

Examples

```
## Not run:
  # while setup of a package
  use_ui()

## End(Not run)
```

Index