# Package: printy (via r-universe)

August 28, 2024

**Title** Helper functions for pretty-printing numbers

**Version** 0.0.0.9005

**Description** This package contains helper functions for formatting numbers.

**Depends** R (>= 4.2.0)

**License** GPL-3 + file LICENSE

**Encoding** UTF-8

**LazyData** true

**Suggests** testthat, pbkrtest, roxygen2

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.1

**Imports** stringr, lme4, dplyr, tidyr, stats, tibble, scales (>= 1.1.0), broom, glue, broom.mixed, rlang (>= 0.1.2), parameters, purrr

**Repository** https://tjmahr.r-universe.dev

**RemoteUrl** https://github.com/tjmahr/printy

**RemoteRef** HEAD

**RemoteSha** b801475d37ee1afc6c4aa8e0ed0580dfe0831d36

## Contents

---

`fmt_effect_md` *Format an effect from a model object in markdown*

---

## Description

Format an effect from a model object in markdown

## Usage

```
fmt_effect_md(
  model,
  effect,
  terms = "besp",
  digits = 2,
  statistic = NULL,
  b_lab = NULL,
  ci_width = 0.95,
  p_value_method = NULL
)
```

## Arguments

| | |
|---|---|
| `model` | a model object |
| `effect` | string naming an effect from a model |
| `terms` | a string representing the terms about the effect to extract and format and the order to print the terms. See details below. Defaults to `"besp"` for parameter estimate, standard error, statistic, *p*-value. |
| `digits` | a vector of digits to use for non-*p*-value terms. Defaults to 2 for 2 decimal places of precision for all terms. This argument can be a vector to set the digits for each term, but in this case, the digits is still ignored for *p*-values. |
| `statistic` | symbol to use for statistic. Defaults to *t* (or *z* in glmer models). |
| `b_lab` | label to print in subscripts after *b* for when `"B"` is one of the terms. |
| `ci_width` | width to use for confidence intervals when the term `"i"` is used. |

## Details

Currently only effects fit by `stats::lm()` and `lme4::lmer()`.

The supported terms are:

- `"b"` - parameter estimate (think b for *beta*)
- `"B"` - parameter estimate with a subscript label provided by `b_lab`
- `"e"` - standard error
- `"s"` - statistic. The symbol for the statistic is set by `statistic`. The default value is `"t"` for a *t*-statistic. Example output: $t = 1$.
- `"S"` - statistic as in `"s"` but with degrees of freedom. Example output: $t(12) = 1$.
- `"i"` - confidence interval. Width is set by `ci_width`.
- `"p"` - *p*-value. The p-value is formatted by `fmt_p_value_md()`.

Degrees of freedom and *p*-values for `lmer()` models use the Kenwood-Rogers approximation provided by `parameters::p_value_kenward()`. This computation can take a while. The confidence-interval calculation uses default confidence interval calculation method used by `broom.mixed::tidy.merMod()`.

## Examples

```
model <- lm(breaks ~ wool * tension, warpbreaks)

# default to: b (beta), e (error), s (statistic), p (p value)
fmt_effect_md(model, "woolB", "besp")

fmt_effect_md(model, "woolB", "Besp", b_lab = "WoolB")

fmt_effect_md(model, "woolB", "i")
```

---

`fmt_fix_digits` *Format a number with a fixed number of digits*

---

## Description

Format a number with a fixed number of digits

## Usage

```
fmt_fix_digits(xs, digits = 2)
```

## Arguments

| | |
|---|---|
| `xs` | a vector of numbers or a character vector representing numbers |
| `digits` | number of digits of precision |

## Examples

```
# what we want to avoid
as.character(round(c(.4001, .1000, .5500), 2))

fmt_fix_digits(c(.4001, .1000, .5500), 1)
fmt_fix_digits(c(.4001, .1000, .5500), 2)
fmt_fix_digits(c(.4001, .1000, .5500), 3)
```

---

`fmt_leading_zero` *Format numbers to remove leading zeros*

---

## Description

Format numbers to remove leading zeros

## Usage

```
fmt_leading_zero(xs)
```

## Arguments

| | |
|---|---|
| `xs` | a vector of numbers or a character vector representing numbers |

## Details

APA format says that values that are bounded between [-1, 1] should not be formatted with a leading zero. Common examples would be correlations, proportions, probabilities and p-values. Why print the digit if it's almost never used?

Zeros are printed to match the precision of the most precise number. For example, `c(0, 0.111)` becomes `c(.000, .111)`

## Value

the vector with leading zeros removed. This function returns a warning if any of the values have an absolute value greater than 1.

## Examples

```
fmt_leading_zero(c(0, 0.111))
fmt_leading_zero(c(0.99, -0.9, -0.0))
```

## fmt_minus_sign *Format negative numbers with a minus sign*

### Description

Format negative numbers with a minus sign

### Usage

```
fmt_minus_sign(xs)
```

### Arguments

| | |
|---|---|
| xs | a vector of numbers or a character vector representing numbers |

### Details

Negative zero `-0`, which might happen from aggressive rounding, does not get a minus sign.

### Value

the vector with leading hyphens replaced with HTML minus signs (`&minus;`).

### Examples

```
fmt_minus_sign(c(1, .2, -1, -.2))

# Don't allow zero to be signed
fmt_minus_sign(c(-0, round(-0.001)))
```

## fmt_p_value *Format a* p*-value*

### Description

Format a *p*-value

### Usage

```
fmt_p_value(xs, digits = 3)
```

### Arguments

| | |
|---|---|
| xs | a vector of numbers or a character vector representing numbers |
| digits | number of digits of precision |

## Value

formatted *-values. Values smaller than the precision `1 / (10 ^ digits)` are replaced with a less than statement `< [precision]`.

## Examples

```
p <- c(1, 0.1, 0.01, 0.001, 0.0001)
fmt_p_value(p, digits = 2)
fmt_p_value(p, digits = 3)
```

---

`fmt_p_value_md` *Format a* p*-value in markdown*

---

## Description

Format a *p*-value in markdown

## Usage

```
fmt_p_value_md(ps)
```

## Arguments

| | |
|---|---|
| `ps` | *p*-values to format |

## Details

`fmt_p_value()` is for formatting p-values with manual precision, but this functions follows some reasonable defaults and returns a markdown formatted string.

Values less than .06 are formatted with 3 digits. Values equal to .06 or greater are formatted with 2 digits.

`scales::label_pvalue()` does the initial rounding and formatting. Then this function strips off the leading 0 of the *p* value.

## Value

a character vector of markdown formatted *p*-values

## Examples

```
fmt_p_value_md(0.0912)
fmt_p_value_md(0.0512)
fmt_p_value_md(0.005)

# "p less than" notation kicks in below .001.
fmt_p_value_md(0.0005)
```

## fmt_remove_html_entities *Replace HTML entities used by this package with UTF-8 codes*

### Description

Replace HTML entities used by this package with UTF-8 codes

### Usage

```
fmt_remove_html_entities(xs)
```

### Arguments

| | |
|---|---|
| xs | a character vector |

### Value

the updated character vector

### Examples

```
x <- "a < &minus;12" |>
  fmt_remove_html_entities()
x
charToRaw(x)
charToRaw("a < -12")

fmt_remove_html_entities("1&ndash;2")
```

## fmt_replace_na *Replace NAs with another value*

### Description

Replace NAs with another value

### Usage

```
fmt_replace_na(xs, replacement = "")
```

### Arguments

| | |
|---|---|
| x | a character vector |

### Value

the updated vector

`skel_conf_interval` *Skeleton for a confidence interval*

## Description

`skel_conf_interval()` is a vectorized function. Use it to make multiple intervals from, say, data-frame columns. `skel_conf_interval_pair()` is the unvectorized function. Use it to make a single interval from a vector (pair) of two numbers.

## Usage

```
skel_conf_interval(xs, ys, skeleton = "[{xs}, {ys}]")

skel_conf_interval_pair(x, skeleton = "[{x[1]}, {x[2]}]")
```

## Arguments

| | |
|---|---|
| `xs` | a vector of the first elements in the intervals |
| `ys` | a vector of the second elements in the intervals |
| `skeleton` | glue-style format to fill. defaults to `"[{xs}, {ys}]"` for `skel_conf_interval()` and `"[{x[1]}, {x[2]}]"` for `skel_conf_interval_pair()`. |
| `x` | a vector of two elements to plug into the confidence interval |

## Details

These functions are wrappers around calls to `glue::glue()`.

Originally, `skel_conf_interval()` was named `skel_conf_interval_v()`.

## Value

strings representing confidence intervals

## Examples

```
skel_conf_interval(c(.1, .2), c(.3, .4))
skel_conf_interval_pair(c(.1, .3))
```

skel_range *Skeleton for a range of numbers*

## Description

skel_range() is a vectorized function. Use it to make multiple range from, say, data-frame columns. skel_range_pair() is the unvectorized function. Use it to make a single range from a vector (pair) of two numbers.

skel_range() is a vectorized function. Use it to make multiple range from, say, data-frame columns. skel_range_pair() is the unvectorized function. Use it to make a single range from a vector (pair) of two numbers.

## Usage

```
skel_range_pair(x, skeleton = "{x[1]}&ndash;{x[2]}")

skel_range(xs, ys, skeleton = "{xs}&ndash;{ys}")
```

## Arguments

| | |
|---|---|
| x | a vector of two elements to plug into the range |
| skeleton | glue-style format to fill. defaults to "{xs}–{ys}" for skel_range() and "{x[1]}–{x[2]}" for skel_range_pair(). |
| xs | a vector of the first elements in the range |
| ys | a vector of the second elements in the range |

## Details

These functions are wrappers around calls to glue::glue().

These functions are wrappers around calls to glue::glue().

## Value

strings representing ranges

strings representing ranges

## Examples

```
skel_range(c(.1, .2), c(.3, .4))
skel_range_pair(c(.1, .3))
skel_range(c(.1, .2), c(.3, .4))
skel_range_pair(c(.1, .3))
```

## skel_se *Skeletons for inline stats*

### Description

Skeletons for inline stats

### Usage

```
skel_se(x, skeleton = "SE = {x}")

skel_ci(x, ci_width = "95", skeleton = "{ci_width}% CI = {x}")
```

### Arguments

| | |
|---|---|
| skeleton | glue-style format to fill. defaults to "SE = {x}" for skel_se() and "95% CI = {x}" for skel_ci(). |
| ci_width | width of the confidence interval to report. Defaults to "95". |
| xs | a vector of the values to plug into the skeleton |

### Value

strings with stats plugged in.

## skel_stat_n_value_pair *Skeleton for t-statistic-like functions*

### Description

This skeleton handles formats like t-statistics (t(df) = value) or correlations (r(df) = value).

### Usage

```
skel_stat_n_value_pair(
  x,
  stat = "t",
  skeleton = "{stat}({x[1]}) = {x[2]}"
)
```

### Arguments

| | |
|---|---|
| x | a two-element vector where the first number is the argument to the statistical function and the second is its value. |
| stat | symbol for the statistic. defaults to "t". |
| skeleton | glue-style format to fill. defaults to "{stat}({x[1]}) = {x[2]}". |

### Value

the formatted string

---

`str_replace_same_as_previous` *Replace strings that duplicate the previous string*

---

### Description

The common use of this function to clean up columns in a presentation-quality table.

### Usage

```
str_replace_same_as_previous(string, replacement)
```

### Arguments

| | |
|---|---|
| `string` | a character vector |
| `replacement` | text to use as a replacement for duplicated values |

### Value

a single character vector with immediately repeating items replaced

### Examples

```
str_replace_same_as_previous(
  c("a", "a", "a", "b", "b", "c", "d", "d"),
  ""
)
```

---

`str_tokenize` *Break a string to individual (character) tokens*

---

### Description

The usual job of this function is to break a string into a vector of individual characters, but it can break strings using other separators.

### Usage

```
str_tokenize(string, pattern = NULL)
```

### Arguments

| | |
|---|---|
| string | a character vector of strings to break |
| pattern | pattern to use for splitting. Defaults to NULL so that strings are split into individual characters. |

### Value

a single character vector of the tokens

### Examples

```
str_tokenize(c("abc", "de"))
str_tokenize(c("abc de fg"), " ")
```

---

## super_split *Split a dataframe into a list of (lists of ...) dataframes*

---

### Description

This function is a streamlined, recursive version of split().

### Usage

```
super_split(.data, ...)
```

### Arguments

| | |
|---|---|
| .data | a dataframe |
| ... | (unquoted) names of columns to split by |

### Value

a list of dataframes when splitting by a single variable, a list of lists of dataframes when splitting by 2 variables, and so on.

### Examples

```
# some kind of 2 by 2 design
df <- data.frame(
  x =    c(1, 2, 3, 4, 5, 6, 7, 8),
  time = c(1, 1, 2, 2, 1, 1, 2, 2),
  group = c("a", "a", "a", "a", "b", "b", "b", "b")
)

super_split(df, group)

super_split(df, time)

# split by group and then split each of those by time
super_split(df, group, time)
```

# Index