Package 'infer'

September 6, 2023

Type Package

Title Tidy Statistical Inference

Version 1.0.5

Description The objective of this package is to perform inference using an expressive statistical grammar that coheres with the tidy design framework.

License MIT + file LICENSE

URL https://github.com/tidymodels/infer, https://infer.tidymodels.org/

BugReports https://github.com/tidymodels/infer/issues

Depends R (>= 3.5.0)

Imports broom, dplyr (>= 0.7.0), generics, ggplot2, glue (>= 1.3.0), grDevices, magrittr, methods, patchwork, purrr, rlang (>= 0.2.0), tibble, tidyr, vctrs

Suggests covr, devtools (>= 1.12.0), fs, knitr, nycflights13, parsnip, rmarkdown, stringr, testthat (>= 3.0.0), vdiffr (>= 1.0.0)

VignetteBuilder knitr

Config/Needs/website tidyverse/tidytemplate

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Config/testthat/edition 3

NeedsCompilation no

Author Andrew Bray [aut],
Chester Ismay [aut] (<https://orcid.org/0000-0003-2820-2547>),
Evgeni Chasnovski [aut] (<https://orcid.org/0000-0002-1617-4019>),
Simon Couch [aut, cre] (<https://orcid.org/0000-0001-5676-5107>),
Ben Baumer [aut] (<https://orcid.org/0000-0002-3279-0516>),
Mine Cetinkaya-Rundel [aut] (<https://orcid.org/0000-0001-6452-2420>),
Ted Laderas [ctb] (<https://orcid.org/0000-0002-6207-7068>),
Nick Solomon [ctb],

Johanna Hardin [ctb],
Albert Y. Kim [ctb] (<https://orcid.org/0000-0001-7824-306X>),
Neal Fultz [ctb],
Doug Friedman [ctb],
Richie Cotton [ctb] (<https://orcid.org/0000-0003-2504-802X>),
Brian Fannin [ctb]

Maintainer Simon Couch <simon.couch@posit.co>

Repository CRAN

Date/Publication 2023-09-06 02:20:02 UTC

R topics documented:

`assume` *Define a theoretical distribution*

Description

This function allows the user to define a null distribution based on theoretical methods. In many infer pipelines, `assume()` can be used in place of `generate()` and `calculate()` to create a null distribution. Rather than outputting a data frame containing a distribution of test statistics calculated from resamples of the observed data, `assume()` outputs a more abstract type of object just containing the distributional details supplied in the `distribution` and `df` arguments. However, `assume()` output can be passed to `visualize()`, `get_p_value()`, and `get_confidence_interval()` in the same way that simulation-based distributions can.

To define a theoretical null distribution (for use in hypothesis testing), be sure to provide a null hypothesis via `hypothesize()`. To define a theoretical sampling distribution (for use in confidence intervals), provide the output of `specify()`. Sampling distributions (only implemented for `t` and `z`) lie on the scale of the data, and will be recentered and rescaled to match the corresponding `stat` given in `calculate()` to calculate the observed statistic.

Usage

```
assume(x, distribution, df = NULL, ...)
```

Arguments

`x`	The output of `specify()` or `hypothesize()`, giving the observed data, variable(s) of interest, and (optionally) null hypothesis.
`distribution`	The distribution in question, as a string. One of `"F"`, `"Chisq"`, `"t"`, or `"z"`.
`df`	Optional. The degrees of freedom parameter(s) for the `distribution` supplied, as a numeric vector. For `distribution = "F"`, this should have length two (e.g. `c(10, 3)`). For `distribution = "Chisq"` or `distribution = "t"`, this should have length one. For `distribution = "z"`, this argument is not required. The package will supply a message if the supplied `df` argument is different from recognized values. See the Details section below for more information.
`...`	Currently ignored.

Details

Note that the assumption being expressed here, for use in theory-based inference, only extends to *distributional* assumptions: the null distribution in question and its parameters. Statistical inference with infer, whether carried out via simulation (i.e. based on pipelines using `generate()` and `calculate()`) or theory (i.e. with `assume()`), always involves the condition that observations are independent of each other.

`infer` only supports theoretical tests on one or two means via the `t` distribution and one or two proportions via the `z`.

For tests comparing two means, if `n1` is the group size for one level of the explanatory variable, and `n2` is that for the other level, `infer` will recognize the following degrees of freedom (`df`) arguments:

- `min(n1 - 1, n2 - 1)`
- `n1 + n2 - 2`
- The `"parameter"` entry of the analogous `stats::t.test()` call

- The "parameter" entry of the analogous stats::t.test() call with var.equal = TRUE

By default, the package will use the "parameter" entry of the analogous stats::t.test() call with var.equal = FALSE (the default).

Value

An infer theoretical distribution that can be passed to helpers like visualize(), get_p_value(), and get_confidence_interval().

Examples

```
# construct theoretical distributions ---------------------------------

# F distribution
# with the `partyid` explanatory variable
gss %>%
  specify(age ~ partyid) %>%
  assume(distribution = "F")

# Chi-squared goodness of fit distribution
# on the `finrela` variable
gss %>%
  specify(response = finrela) %>%
  hypothesize(null = "point",
              p = c("far below average" = 1/6,
                    "below average" = 1/6,
                    "average" = 1/6,
                    "above average" = 1/6,
                    "far above average" = 1/6,
                    "DK" = 1/6)) %>%
  assume("Chisq")

# Chi-squared test of independence
# on the `finrela` and `sex` variables
gss %>%
  specify(formula = finrela ~ sex) %>%
  assume(distribution = "Chisq")

# T distribution
gss %>%
  specify(age ~ college) %>%
  assume("t")

# Z distribution
gss %>%
  specify(response = sex, success = "female") %>%
  assume("z")

## Not run:
# each of these distributions can be passed to infer helper
# functions alongside observed statistics!
```

```
# for example, a 1-sample t-test -------------------------------------

# calculate the observed statistic
obs_stat <- gss %>%
  specify(response = hours) %>%
  hypothesize(null = "point", mu = 40) %>%
  calculate(stat = "t")

# construct a null distribution
null_dist <- gss %>%
  specify(response = hours) %>%
  assume("t")

# juxtapose them visually
visualize(null_dist) +
  shade_p_value(obs_stat, direction = "both")

# calculate a p-value
get_p_value(null_dist, obs_stat, direction = "both")

# or, an F test ------------------------------------------------------

# calculate the observed statistic
obs_stat <- gss %>%
  specify(age ~ partyid) %>%
  hypothesize(null = "independence") %>%
  calculate(stat = "F")

# construct a null distribution
null_dist <- gss %>%
  specify(age ~ partyid) %>%
  assume(distribution = "F")

# juxtapose them visually
visualize(null_dist) +
  shade_p_value(obs_stat, direction = "both")

# calculate a p-value
get_p_value(null_dist, obs_stat, direction = "both")

## End(Not run)
```

`calculate` *Calculate summary statistics*

Description

Given the output of `specify()` and/or `hypothesize()`, this function will return the observed statistic specified with the `stat` argument. Some test statistics, such as `Chisq`, `t`, and `z`, require a null

hypothesis. If provided the output of generate(), the function will calculate the supplied stat for each replicate.

Learn more in vignette("infer").

Usage

```
calculate(
  x,
  stat = c("mean", "median", "sum", "sd", "prop", "count", "diff in means",
   "diff in medians", "diff in props", "Chisq", "F", "slope", "correlation", "t", "z",
    "ratio of props", "odds ratio", "ratio of means"),
  order = NULL,
  ...
)
```

Arguments

x	The output from generate() for computation-based inference or the output from hypothesize() piped in to here for theory-based inference.
stat	A string giving the type of the statistic to calculate. Current options include "mean", "median", "sum", "sd", "prop", "count", "diff in means", "diff in medians", "diff in props", "Chisq" (or "chisq"), "F" (or "f"), "t", "z", "ratio of props", "slope", "odds ratio", "ratio of means", and "correlation". infer only supports theoretical tests on one or two means via the "t" distribution and one or two proportions via the "z".
order	A string vector of specifying the order in which the levels of the explanatory variable should be ordered for subtraction (or division for ratio-based statistics), where order = c("first", "second") means ("first" - "second"), or the analogue for ratios. Needed for inference on difference in means, medians, proportions, ratios, t, and z statistics.
...	To pass options like na.rm = TRUE into functions like mean(), sd(), etc. Can also be used to supply hypothesized null values for the "t" statistic or additional arguments to stats::chisq.test().

Value

A tibble containing a stat column of calculated statistics.

Missing levels in small samples

In some cases, when bootstrapping with small samples, some generated bootstrap samples will have only one level of the explanatory variable present. For some test statistics, the calculated statistic in these cases will be NaN. The package will omit non-finite values from visualizations (with a warning) and raise an error in p-value calculations.

Reproducibility

When using the infer package for research, or in other cases when exact reproducibility is a priority, be sure the set the seed for R's random number generator. infer will respect the random seed specified in the `set.seed()` function, returning the same result when `generate()`ing data given an identical seed. For instance, we can calculate the difference in mean `age` by `college` degree status using the `gss` dataset from 10 versions of the `gss` resampled with permutation using the following code.

```
set.seed(1)

gss %>%
  specify(age ~ college) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 5, type = "permute") %>%
  calculate("diff in means", order = c("degree", "no degree"))
```

```
## Response: age (numeric)
## Explanatory: college (factor)
## Null Hypothesis: independence
## # A tibble: 5 x 2
##   replicate   stat
##       <int>  <dbl>
## 1         1 -0.531
## 2         2 -2.35
## 3         3  0.764
## 4         4  0.280
## 5         5  0.350
```

Setting the seed to the same value again and rerunning the same code will produce the same result.

```
# set the seed
set.seed(1)

gss %>%
  specify(age ~ college) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 5, type = "permute") %>%
  calculate("diff in means", order = c("degree", "no degree"))
```

```
## Response: age (numeric)
## Explanatory: college (factor)
## Null Hypothesis: independence
## # A tibble: 5 x 2
##   replicate   stat
##       <int>  <dbl>
## 1         1 -0.531
## 2         2 -2.35
## 3         3  0.764
```

```
## 4          4  0.280
## 5          5  0.350
```

Please keep this in mind when writing infer code that utilizes resampling with `generate()`.

See Also

`visualize()`, `get_p_value()`, and `get_confidence_interval()` to extract value from this function's outputs.

Other core functions: `generate()`, `hypothesize()`, `specify()`

Examples

```
# calculate a null distribution of hours worked per week under
# the null hypothesis that the mean is 40
gss %>%
  specify(response = hours) %>%
  hypothesize(null = "point", mu = 40) %>%
  generate(reps = 200, type = "bootstrap") %>%
  calculate(stat = "mean")

# calculate the corresponding observed statistic
gss %>%
  specify(response = hours) %>%
  calculate(stat = "mean")

# calculate a null distribution assuming independence between age
# of respondent and whether they have a college degree
gss %>%
  specify(age ~ college) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 200, type = "permute") %>%
  calculate("diff in means", order = c("degree", "no degree"))

# calculate the corresponding observed statistic
gss %>%
  specify(age ~ college) %>%
  calculate("diff in means", order = c("degree", "no degree"))

# some statistics require a null hypothesis
 gss %>%
   specify(response = hours) %>%
   hypothesize(null = "point", mu = 40) %>%
   calculate(stat = "t")

# more in-depth explanation of how to use the infer package
## Not run:
vignette("infer")

## End(Not run)
```

chisq_stat *Tidy chi-squared test statistic*

Description

@description

Usage

```
chisq_stat(x, formula, response = NULL, explanatory = NULL, ...)
```

Arguments

x	A data frame that can be coerced into a tibble.
formula	A formula with the response variable on the left and the explanatory on the right.
response	The variable name in x that will serve as the response. This is alternative to using the formula argument.
explanatory	The variable name in x that will serve as the explanatory variable.
...	Additional arguments for chisq.test().

Details

A shortcut wrapper function to get the observed test statistic for a chisq test. Uses chisq.test(), which applies a continuity correction. This function has been deprecated in favor of the more general observe().

See Also

Other wrapper functions: chisq_test(), observe(), prop_test(), t_stat(), t_test()

Other functions for calculating observed statistics: observe(), t_stat()

Examples

```
# chi-squared test statistic for test of independence
# of college completion status depending and one's
# self-identified income class
chisq_stat(gss, college ~ finrela)

# chi-squared test statistic for a goodness of fit
# test on whether self-identified income class
# follows a uniform distribution
chisq_stat(gss,
           response = finrela,
           p = c("far below average" = 1/6,
                 "below average" = 1/6,
                 "average" = 1/6,
                 "above average" = 1/6,
```

```
                  "far above average" = 1/6,
                  "DK" = 1/6))
```

`chisq_test` *Tidy chi-squared test*

Description

A tidier version of chisq.test() for goodness of fit tests and tests of independence.

Usage

```
chisq_test(x, formula, response = NULL, explanatory = NULL, ...)
```

Arguments

`x`	A data frame that can be coerced into a tibble.
`formula`	A formula with the response variable on the left and the explanatory on the right.
`response`	The variable name in `x` that will serve as the response. This is alternative to using the `formula` argument.
`explanatory`	The variable name in `x` that will serve as the explanatory variable.
`...`	Additional arguments for chisq.test().

See Also

Other wrapper functions: `chisq_stat()`, `observe()`, `prop_test()`, `t_stat()`, `t_test()`

Examples

```
# chi-squared test of independence for college completion
# status depending on one's self-identified income class
chisq_test(gss, college ~ finrela)

# chi-squared goodness of fit test on whether self-identified
# income class follows a uniform distribution
chisq_test(gss,
           response = finrela,
           p = c("far below average" = 1/6,
                 "below average" = 1/6,
                 "average" = 1/6,
                 "above average" = 1/6,
                 "far above average" = 1/6,
                 "DK" = 1/6))
```

deprecated *Deprecated functions and objects*

Description

These functions and objects should no longer be used. They will be removed in a future release of `infer`.

Usage

```
conf_int(x, level = 0.95, type = "percentile", point_estimate = NULL)

p_value(x, obs_stat, direction)
```

Arguments

`x`	See the non-deprecated function.
`level`	See the non-deprecated function.
`type`	See the non-deprecated function.
`point_estimate`	See the non-deprecated function.
`obs_stat`	See the non-deprecated function.
`direction`	See the non-deprecated function.

See Also

`get_p_value()`, `get_confidence_interval()`, `generate()`

fit.infer *Fit linear models to infer objects*

Description

Given the output of an infer core function, this function will fit a linear model using `stats::glm()` according to the formula and data supplied earlier in the pipeline. If passed the output of `specify()` or `hypothesize()`, the function will fit one model. If passed the output of `generate()`, it will fit a model to each data resample, denoted in the `replicate` column. The family of the fitted model depends on the type of the response variable. If the response is numeric, `fit()` will use `family = "gaussian"` (linear regression). If the response is a 2-level factor or character, `fit()` will use `family = "binomial"` (logistic regression). To fit character or factor response variables with more than two levels, we recommend `parsnip::multinom_reg()`.

infer provides a fit "method" for infer objects, which is a way of carrying out model fitting as applied to infer output. The "generic," imported from the generics package and re-exported from this package, provides the general form of `fit()` that points to infer's method when called on an infer object. That generic is also documented here.

Learn more in `vignette("infer")`.

Usage

```
## S3 method for class 'infer'
fit(object, ...)
```

Arguments

`object`	Output from an infer function—likely `generate()` or `specify()`—which specifies the formula and data to fit a model to.
`...`	Any optional arguments to pass along to the model fitting function. See `stats::glm()` for more information.

Details

Randomization-based statistical inference with multiple explanatory variables requires careful consideration of the null hypothesis in question and its implications for permutation procedures. Inference for partial regression coefficients via the permutation method implemented in `generate()` for multiple explanatory variables, consistent with its meaning elsewhere in the package, is subject to additional distributional assumptions beyond those required for one explanatory variable. Namely, the distribution of the response variable must be similar to the distribution of the errors under the null hypothesis' specification of a fixed effect of the explanatory variables. (This null hypothesis is reflected in the `variables` argument to `generate()`. By default, all of the explanatory variables are treated as fixed.) A general rule of thumb here is, if there are large outliers in the distributions of any of the explanatory variables, this distributional assumption will not be satisfied; when the response variable is permuted, the (presumably outlying) value of the response will no longer be paired with the outlier in the explanatory variable, causing an outsize effect on the resulting slope coefficient for that explanatory variable.

More sophisticated methods that are outside of the scope of this package requiring fewer—or less strict—distributional assumptions exist. For an overview, see "Permutation tests for univariate or multivariate analysis of variance and regression" (Marti J. Anderson, 2001), doi:10.1139/cjfas583-626.

Value

A tibble containing the following columns:

- `replicate`: Only supplied if the input object had been previously passed to `generate()`. A number corresponding to which resample of the original data set the model was fitted to.
- `term`: The explanatory variable (or intercept) in question.
- `estimate`: The model coefficient for the given resample (`replicate`) and explanatory variable (`term`).

Reproducibility

When using the infer package for research, or in other cases when exact reproducibility is a priority, be sure the set the seed for R's random number generator. infer will respect the random seed specified in the `set.seed()` function, returning the same result when `generate()`ing data given an identical seed. For instance, we can calculate the difference in mean `age` by `college` degree status

using the gss dataset from 10 versions of the gss resampled with permutation using the following code.

```
set.seed(1)

gss %>%
  specify(age ~ college) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 5, type = "permute") %>%
  calculate("diff in means", order = c("degree", "no degree"))

## Response: age (numeric)
## Explanatory: college (factor)
## Null Hypothesis: independence
## # A tibble: 5 x 2
##   replicate   stat
##       <int>  <dbl>
## 1         1 -0.531
## 2         2 -2.35
## 3         3  0.764
## 4         4  0.280
## 5         5  0.350
```

Setting the seed to the same value again and rerunning the same code will produce the same result.

```
# set the seed
set.seed(1)

gss %>%
  specify(age ~ college) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 5, type = "permute") %>%
  calculate("diff in means", order = c("degree", "no degree"))

## Response: age (numeric)
## Explanatory: college (factor)
## Null Hypothesis: independence
## # A tibble: 5 x 2
##   replicate   stat
##       <int>  <dbl>
## 1         1 -0.531
## 2         2 -2.35
## 3         3  0.764
## 4         4  0.280
## 5         5  0.350
```

Please keep this in mind when writing infer code that utilizes resampling with `generate()`.

Examples

```
# fit a linear model predicting number of hours worked per
# week using respondent age and degree status.
observed_fit <- gss %>%
  specify(hours ~ age + college) %>%
  fit()

observed_fit

# fit 100 models to resamples of the gss dataset, where the response
# `hours` is permuted in each. note that this code is the same as
# the above except for the addition of the `generate` step.
null_fits <- gss %>%
  specify(hours ~ age + college) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 100, type = "permute") %>%
  fit()

null_fits

# for logistic regression, just supply a binary response variable!
# (this can also be made explicit via the `family` argument in ...)
gss %>%
  specify(college ~ age + hours) %>%
  fit()

# more in-depth explanation of how to use the infer package
## Not run:
vignette("infer")

## End(Not run)
```

`generate` *Generate resamples, permutations, or simulations*

Description

Generation creates a simulated distribution from `specify()`. In the context of confidence intervals, this is a bootstrap distribution based on the result of `specify()`. In the context of hypothesis testing, this is a null distribution based on the result of `specify()` and `hypothesize()`.

Learn more in `vignette("infer")`.

Usage

```
generate(x, reps = 1, type = NULL, variables = !!response_expr(x), ...)
```

Arguments

x	A data frame that can be coerced into a tibble.
reps	The number of resamples to generate.
type	The method used to generate resamples of the observed data reflecting the null hypothesis. Currently one of "bootstrap", "permute", or "draw" (see below).
variables	If type = "permute", a set of unquoted column names in the data to permute (independently of each other). Defaults to only the response variable. Note that any derived effects that depend on these columns (e.g., interaction effects) will also be affected.
...	Currently ignored.

Value

A tibble containing reps generated datasets, indicated by the replicate column.

Generation Types

The type argument determines the method used to create the null distribution.

- bootstrap: A bootstrap sample will be drawn for each replicate, where a sample of size equal to the input sample size is drawn (with replacement) from the input sample data.
- permute: For each replicate, each input value will be randomly reassigned (without replacement) to a new output value in the sample.
- draw: A value will be sampled from a theoretical distribution with parameter p specified in hypothesize() for each replicate. This option is currently only applicable for testing on one proportion. This generation type was previously called "simulate", which has been superseded.

Reproducibility

When using the infer package for research, or in other cases when exact reproducibility is a priority, be sure the set the seed for R's random number generator. infer will respect the random seed specified in the set.seed() function, returning the same result when generate()ing data given an identical seed. For instance, we can calculate the difference in mean age by college degree status using the gss dataset from 10 versions of the gss resampled with permutation using the following code.

```
set.seed(1)

gss %>%
  specify(age ~ college) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 5, type = "permute") %>%
  calculate("diff in means", order = c("degree", "no degree"))
```

```
## Response: age (numeric)
## Explanatory: college (factor)
## Null Hypothesis: independence
## # A tibble: 5 x 2
##   replicate   stat
##       <int>  <dbl>
## 1         1 -0.531
## 2         2 -2.35
## 3         3  0.764
## 4         4  0.280
## 5         5  0.350
```

Setting the seed to the same value again and rerunning the same code will produce the same result.

```
# set the seed
set.seed(1)

gss %>%
  specify(age ~ college) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 5, type = "permute") %>%
  calculate("diff in means", order = c("degree", "no degree"))
```

```
## Response: age (numeric)
## Explanatory: college (factor)
## Null Hypothesis: independence
## # A tibble: 5 x 2
##   replicate   stat
##       <int>  <dbl>
## 1         1 -0.531
## 2         2 -2.35
## 3         3  0.764
## 4         4  0.280
## 5         5  0.350
```

Please keep this in mind when writing infer code that utilizes resampling with `generate()`.

See Also

Other core functions: `calculate()`, `hypothesize()`, `specify()`

Examples

```
# generate a null distribution by taking 200 bootstrap samples
gss %>%
 specify(response = hours) %>%
 hypothesize(null = "point", mu = 40) %>%
 generate(reps = 200, type = "bootstrap")
```

```
# generate a null distribution for the independence of
# two variables by permuting their values 200 times
gss %>%
 specify(partyid ~ age) %>%
 hypothesize(null = "independence") %>%
 generate(reps = 200, type = "permute")

# generate a null distribution via sampling from a
# binomial distribution 200 times
gss %>%
specify(response = sex, success = "female") %>%
  hypothesize(null = "point", p = .5) %>%
  generate(reps = 200, type = "draw") %>%
  calculate(stat = "z")

# more in-depth explanation of how to use the infer package
## Not run:
vignette("infer")

## End(Not run)
```

`get_confidence_interval` *Compute confidence interval*

Description

Compute a confidence interval around a summary statistic. Both simulation-based and theoretical methods are supported, though only `type = "se"` is supported for theoretical methods.

Learn more in `vignette("infer")`.

Usage

```
get_confidence_interval(x, level = 0.95, type = NULL, point_estimate = NULL)

get_ci(x, level = 0.95, type = NULL, point_estimate = NULL)
```

Arguments

`x`	A distribution. For simulation-based inference, a data frame containing a distribution of `calculate()`d statistics or `fit()`ted coefficient estimates. This object should have been passed to `generate()` before being supplied or `calculate()` to `fit()`. For theory-based inference, output of `assume()`. Distributions for confidence intervals do not require a null hypothesis via `hypothesize()`.
`level`	A numerical value between 0 and 1 giving the confidence level. Default value is 0.95.

type	A string giving which method should be used for creating the confidence interval. The default is "percentile" with "se" corresponding to (multiplier * standard error) and "bias-corrected" for bias-corrected interval as other options.
point_estimate	A data frame containing the observed statistic (in a calculate()-based workflow) or observed fit (in a fit()-based workflow). This object is likely the output of calculate() or fit() and need not to have been passed to generate(). Set to NULL by default. Must be provided if type is "se" or "bias-corrected".

Details

A null hypothesis is not required to compute a confidence interval. However, including hypothesize() in a pipeline leading to get_confidence_interval() will not break anything. This can be useful when computing a confidence interval using the same distribution used to compute a p-value.

Theoretical confidence intervals (i.e. calculated by supplying the output of assume() to the x argument) require that the point estimate lies on the scale of the data. The distribution defined in assume() will be recentered and rescaled to align with the point estimate, as can be shown in the output of visualize() when paired with shade_confidence_interval(). Confidence intervals are implemented for the following distributions and point estimates:

- distribution = "t": point_estimate should be the output of calculate() with stat = "mean" or stat = "diff in means"
- distribution = "z": point_estimate should be the output of calculate() with stat = "prop" or stat = "diff in props"

Value

A tibble containing the following columns:

- term: The explanatory variable (or intercept) in question. Only supplied if the input had been previously passed to fit().
- lower_ci, upper_ci: The lower and upper bounds of the confidence interval, respectively.

Aliases

get_ci() is an alias of get_confidence_interval(). conf_int() is a deprecated alias of get_confidence_interval().

See Also

Other auxillary functions: get_p_value()

Examples

```
boot_dist <- gss %>%
  # We're interested in the number of hours worked per week
  specify(response = hours) %>%
  # Generate bootstrap samples
  generate(reps = 1000, type = "bootstrap") %>%
```

```
  # Calculate mean of each bootstrap sample
  calculate(stat = "mean")

boot_dist %>%
  # Calculate the confidence interval around the point estimate
  get_confidence_interval(
    # At the 95% confidence level; percentile method
    level = 0.95
  )

# for type = "se" or type = "bias-corrected" we need a point estimate
sample_mean <- gss %>%
  specify(response = hours) %>%
  calculate(stat = "mean")

boot_dist %>%
  get_confidence_interval(
    point_estimate = sample_mean,
    # At the 95% confidence level
    level = 0.95,
    # Using the standard error method
    type = "se"
  )

# using a theoretical distribution ----------------------------------

# define a sampling distribution
sampling_dist <- gss %>%
  specify(response = hours) %>%
  assume("t")

# get the confidence interval---note that the
# point estimate is required here
get_confidence_interval(
  sampling_dist,
  level = .95,
  point_estimate = sample_mean
)

# using a model fitting workflow ------------------------

# fit a linear model predicting number of hours worked per
# week using respondent age and degree status.
observed_fit <- gss %>%
  specify(hours ~ age + college) %>%
  fit()

observed_fit

# fit 100 models to resamples of the gss dataset, where the response
# `hours` is permuted in each. note that this code is the same as
# the above except for the addition of the `generate` step.
null_fits <- gss %>%
```

```
  specify(hours ~ age + college) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 100, type = "permute") %>%
  fit()

null_fits

get_confidence_interval(
  null_fits,
  point_estimate = observed_fit,
  level = .95
)

# more in-depth explanation of how to use the infer package
## Not run:
vignette("infer")

## End(Not run)
```

`get_p_value` *Compute p-value*

Description

Compute a p-value from a null distribution and observed statistic.

Learn more in `vignette("infer")`.

Usage

```
get_p_value(x, obs_stat, direction)

## Default S3 method:
get_p_value(x, obs_stat, direction)

get_pvalue(x, obs_stat, direction)

## S3 method for class 'infer_dist'
get_p_value(x, obs_stat, direction)
```

Arguments

x	A null distribution. For simulation-based inference, a data frame containing a distribution of `calculate()`d statistics or `fit()`ted coefficient estimates. This object should have been passed to `generate()` before being supplied or `calculate()` to `fit()`. For theory-based inference, the output of `assume()`.
obs_stat	A data frame containing the observed statistic (in a `calculate()`-based workflow) or observed fit (in a `fit()`-based workflow). This object is likely the output of `calculate()` or `fit()` and need not to have been passed to `generate()`.

direction — A character string. Options are "less", "greater", or "two-sided". Can also use "left", "right", "both", "two_sided", or "two sided", "two.sided".

Value

A tibble containing the following columns:

- term: The explanatory variable (or intercept) in question. Only supplied if the input had been previously passed to fit().
- p_value: A value in [0, 1] giving the probability that a statistic/coefficient as or more extreme than the observed statistic/coefficient would occur if the null hypothesis were true.

Aliases

get_pvalue() is an alias of get_p_value(). p_value is a deprecated alias of get_p_value().

Zero p-value

Though a true p-value of 0 is impossible, get_p_value() may return 0 in some cases. This is due to the simulation-based nature of the {infer} package; the output of this function is an approximation based on the number of reps chosen in the generate() step. When the observed statistic is very unlikely given the null hypothesis, and only a small number of reps have been generated to form a null distribution, it is possible that the observed statistic will be more extreme than every test statistic generated to form the null distribution, resulting in an approximate p-value of 0. In this case, the true p-value is a small value likely less than 3/reps (based on a poisson approximation).

In the case that a p-value of zero is reported, a warning message will be raised to caution the user against reporting a p-value exactly equal to 0.

See Also

Other auxillary functions: get_confidence_interval()

Examples

```
# using a simulation-based null distribution ------------------------------

# find the point estimate---mean number of hours worked per week
point_estimate <- gss %>%
  specify(response = hours) %>%
  calculate(stat = "mean")

# starting with the gss dataset
gss %>%
  # ...we're interested in the number of hours worked per week
  specify(response = hours) %>%
  # hypothesizing that the mean is 40
  hypothesize(null = "point", mu = 40) %>%
  # generating data points for a null distribution
  generate(reps = 1000, type = "bootstrap") %>%
  # finding the null distribution
```

```
  calculate(stat = "mean") %>%
  get_p_value(obs_stat = point_estimate, direction = "two-sided")

# using a theoretical null distribution ----------------------------------

# calculate the observed statistic
obs_stat <- gss %>%
  specify(response = hours) %>%
  hypothesize(null = "point", mu = 40) %>%
  calculate(stat = "t")

# define a null distribution
null_dist <- gss %>%
  specify(response = hours) %>%
  assume("t")

# calculate a p-value
get_p_value(null_dist, obs_stat, direction = "both")

# using a model fitting workflow -----------------------------------------

# fit a linear model predicting number of hours worked per
# week using respondent age and degree status.
observed_fit <- gss %>%
  specify(hours ~ age + college) %>%
  fit()

observed_fit

# fit 100 models to resamples of the gss dataset, where the response
# `hours` is permuted in each. note that this code is the same as
# the above except for the addition of the `generate` step.
null_fits <- gss %>%
  specify(hours ~ age + college) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 100, type = "permute") %>%
  fit()

null_fits

get_p_value(null_fits, obs_stat = observed_fit, direction = "two-sided")

# more in-depth explanation of how to use the infer package
## Not run:
vignette("infer")

## End(Not run)
```

---

gss *Subset of data from the General Social Survey (GSS).*

Description

The General Social Survey is a high-quality survey which gathers data on American society and opinions, conducted since 1972. This data set is a sample of 500 entries from the GSS, spanning years 1973-2018, including demographic markers and some economic variables. Note that this data is included for demonstration only, and should not be assumed to provide accurate estimates relating to the GSS. However, due to the high quality of the GSS, the unweighted data will approximate the weighted data in some analyses.

Usage

```
gss
```

Format

A tibble with 500 rows and 11 variables:

year year respondent was surveyed

age age at time of survey, truncated at 89

sex respondent's sex (self-identified)

college whether on not respondent has a college degree, including junior/community college

partyid political party affiliation

hompop number of persons in household

hours number of hours worked in week before survey, truncated at 89

income total family income

class subjective socioeconomic class identification

finrela opinion of family income

weight survey weight

Source

https://gss.norc.org

---

hypothesize *Declare a null hypothesis*

Description

Declare a null hypothesis about variables selected in `specify()`.

Learn more in `vignette("infer")`.

Usage

```
hypothesize(x, null, p = NULL, mu = NULL, med = NULL, sigma = NULL)

hypothesise(x, null, p = NULL, mu = NULL, med = NULL, sigma = NULL)
```

Arguments

x	A data frame that can be coerced into a tibble.
null	The null hypothesis. Options include "independence", "point", and "paired independence". • independence: Should be used with both a response and explanatory variable. Indicates that the values of the specified response variable are independent of the associated values in explanatory. • point: Should be used with only a response variable. Indicates that a point estimate based on the values in response is associated with a parameter. Sometimes requires supplying one of p, mu, med, or sigma. • paired independence: Should be used with only a response variable giving the pre-computed difference between paired observations. Indicates that the order of subtraction between paired values does not affect the resulting distribution.
p	The true proportion of successes (a number between 0 and 1). To be used with point null hypotheses when the specified response variable is categorical.
mu	The true mean (any numerical value). To be used with point null hypotheses when the specified response variable is continuous.
med	The true median (any numerical value). To be used with point null hypotheses when the specified response variable is continuous.
sigma	The true standard deviation (any numerical value). To be used with point null hypotheses.

Value

A tibble containing the response (and explanatory, if specified) variable data with parameter information stored as well.

See Also

Other core functions: calculate(), generate(), specify()

Examples

```
# hypothesize independence of two variables
gss %>%
 specify(college ~ partyid, success = "degree") %>%
 hypothesize(null = "independence")

# hypothesize a mean number of hours worked per week of 40
gss %>%
  specify(response = hours) %>%
  hypothesize(null = "point", mu = 40)

# more in-depth explanation of how to use the infer package
## Not run:
vignette("infer")
```

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

`infer` *infer: a grammar for statistical inference*

Description

The objective of this package is to perform statistical inference using a grammar that illustrates the underlying concepts and a format that coheres with the tidyverse.

Details

For an overview of how to use the core functionality, see `vignette("infer")`

Author(s)

Maintainer: Simon Couch <simon.couch@posit.co> (ORCID)

Authors:

- Andrew Bray <abray@reed.edu>
- Chester Ismay <chester.ismay@gmail.com> (ORCID)
- Evgeni Chasnovski <evgeni.chasnovski@gmail.com> (ORCID)
- Ben Baumer <ben.baumer@gmail.com> (ORCID)
- Mine Cetinkaya-Rundel <mine@stat.duke.edu> (ORCID)

Other contributors:

- Ted Laderas <tedladeras@gmail.com> (ORCID) [contributor]
- Nick Solomon <nick.solomon@datacamp.com> [contributor]
- Johanna Hardin <Jo.Hardin@pomona.edu> [contributor]
- Albert Y. Kim <albert.ys.kim@gmail.com> (ORCID) [contributor]
- Neal Fultz <nfultz@gmail.com> [contributor]
- Doug Friedman <doug.nhp@gmail.com> [contributor]
- Richie Cotton <richie@datacamp.com> (ORCID) [contributor]
- Brian Fannin <captain@pirategrunt.com> [contributor]

See Also

Useful links:

- https://github.com/tidymodels/infer
- https://infer.tidymodels.org/
- Report bugs at https://github.com/tidymodels/infer/issues

observe *Calculate observed statistics*

Description

This function is a wrapper that calls specify(), hypothesize(), and calculate() consecutively that can be used to calculate observed statistics from data. hypothesize() will only be called if a point null hypothesis parameter is supplied.

Learn more in vignette("infer").

Usage

```
observe(
  x,
  formula,
  response = NULL,
  explanatory = NULL,
  success = NULL,
  null = NULL,
  p = NULL,
  mu = NULL,
  med = NULL,
  sigma = NULL,
  stat = c("mean", "median", "sum", "sd", "prop", "count", "diff in means",
   "diff in medians", "diff in props", "Chisq", "F", "slope", "correlation", "t", "z",
    "ratio of props", "odds ratio"),
  order = NULL,
  ...
)
```

Arguments

x	A data frame that can be coerced into a tibble.
formula	A formula with the response variable on the left and the explanatory on the right. Alternatively, a response and explanatory argument can be supplied.
response	The variable name in x that will serve as the response. This is an alternative to using the formula argument.
explanatory	The variable name in x that will serve as the explanatory variable. This is an alternative to using the formula argument.
success	The level of response that will be considered a success, as a string. Needed for inference on one proportion, a difference in proportions, and corresponding z stats.
null	The null hypothesis. Options include "independence", "point", and "paired independence".

- `independence`: Should be used with both a `response` and `explanatory` variable. Indicates that the values of the specified `response` variable are independent of the associated values in `explanatory`.
- `point`: Should be used with only a `response` variable. Indicates that a point estimate based on the values in `response` is associated with a parameter. Sometimes requires supplying one of `p`, `mu`, `med`, or `sigma`.
- `paired independence`: Should be used with only a `response` variable giving the pre-computed difference between paired observations. Indicates that the order of subtraction between paired values does not affect the resulting distribution.

`p` The true proportion of successes (a number between 0 and 1). To be used with point null hypotheses when the specified response variable is categorical.

`mu` The true mean (any numerical value). To be used with point null hypotheses when the specified response variable is continuous.

`med` The true median (any numerical value). To be used with point null hypotheses when the specified response variable is continuous.

`sigma` The true standard deviation (any numerical value). To be used with point null hypotheses.

`stat` A string giving the type of the statistic to calculate. Current options include `"mean"`, `"median"`, `"sum"`, `"sd"`, `"prop"`, `"count"`, `"diff in means"`, `"diff in medians"`, `"diff in props"`, `"Chisq"` (or `"chisq"`), `"F"` (or `"f"`), `"t"`, `"z"`, `"ratio of props"`, `"slope"`, `"odds ratio"`, `"ratio of means"`, and `"correlation"`. `infer` only supports theoretical tests on one or two means via the `"t"` distribution and one or two proportions via the `"z"`.

`order` A string vector of specifying the order in which the levels of the explanatory variable should be ordered for subtraction (or division for ratio-based statistics), where `order = c("first", "second")` means `("first" - "second")`, or the analogue for ratios. Needed for inference on difference in means, medians, proportions, ratios, t, and z statistics.

`...` To pass options like `na.rm = TRUE` into functions like mean(), sd(), etc. Can also be used to supply hypothesized null values for the `"t"` statistic or additional arguments to `stats::chisq.test()`.

Value

A 1-column tibble containing the calculated statistic `stat`.

See Also

Other wrapper functions: `chisq_stat()`, `chisq_test()`, `prop_test()`, `t_stat()`, `t_test()`

Other functions for calculating observed statistics: `chisq_stat()`, `t_stat()`

Examples

```
# calculating the observed mean number of hours worked per week
gss %>%
  observe(hours ~ NULL, stat = "mean")
```

```
# equivalently, calculating the same statistic with the core verbs
gss %>%
  specify(response = hours) %>%
  calculate(stat = "mean")

# calculating a t statistic for hypothesized mu = 40 hours worked/week
gss %>%
  observe(hours ~ NULL, stat = "t", null = "point", mu = 40)

# equivalently, calculating the same statistic with the core verbs
gss %>%
  specify(response = hours) %>%
  hypothesize(null = "point", mu = 40) %>%
  calculate(stat = "t")

# similarly for a difference in means in age based on whether
# the respondent has a college degree
observe(
  gss,
  age ~ college,
  stat = "diff in means",
  order = c("degree", "no degree")
)

# equivalently, calculating the same statistic with the core verbs
gss %>%
  specify(age ~ college) %>%
  calculate("diff in means", order = c("degree", "no degree"))

# for a more in-depth explanation of how to use the infer package
## Not run:
vignette("infer")

## End(Not run)
```

`print.infer` *Print methods*

Description

Print methods

Usage

```
## S3 method for class 'infer'
print(x, ...)

## S3 method for class 'infer_layer'
```

```
print(x, ...)

## S3 method for class 'infer_dist'
print(x, ...)
```

Arguments

x	An object of class `infer`, i.e. output from `specify()` or `hypothesize()`, or of class `infer_layer`, i.e. output from `shade_p_value()` or `shade_confidence_interval()`.
...	Arguments passed to methods.

---

prop_test *Tidy proportion test*

Description

A tidier version of prop.test() for equal or given proportions.

Usage

```
prop_test(
  x,
  formula,
  response = NULL,
  explanatory = NULL,
  p = NULL,
  order = NULL,
  alternative = "two-sided",
  conf_int = TRUE,
  conf_level = 0.95,
  success = NULL,
  correct = NULL,
  z = FALSE,
  ...
)
```

Arguments

x	A data frame that can be coerced into a tibble.
formula	A formula with the response variable on the left and the explanatory on the right, where an explanatory variable NULL indicates a test of a single proportion.
response	The variable name in `x` that will serve as the response. This is alternative to using the `formula` argument. This is an alternative to the formula interface.
explanatory	The variable name in `x` that will serve as the explanatory variable. Optional. This is an alternative to the formula interface.

p	A numeric vector giving the hypothesized null proportion of success for each group.
order	A string vector specifying the order in which the proportions should be subtracted, where order = c("first", "second") means "first" - "second". Ignored for one-sample tests, and optional for two sample tests.
alternative	Character string giving the direction of the alternative hypothesis. Options are "two-sided" (default), "greater", or "less". Only used when testing the null that a single proportion equals a given value, or that two proportions are equal; ignored otherwise.
conf_int	A logical value for whether to report the confidence interval or not. TRUE by default, ignored if p is specified for a two-sample test. Only used when testing the null that a single proportion equals a given value, or that two proportions are equal; ignored otherwise.
conf_level	A numeric value between 0 and 1. Default value is 0.95. Only used when testing the null that a single proportion equals a given value, or that two proportions are equal; ignored otherwise.
success	The level of response that will be considered a success, as a string. Only used when testing the null that a single proportion equals a given value, or that two proportions are equal; ignored otherwise.
correct	A logical indicating whether Yates' continuity correction should be applied where possible. If z = TRUE, the correct argument will be overwritten as FALSE. Otherwise defaults to correct = TRUE.
z	A logical value for whether to report the statistic as a standard normal deviate or a Pearson's chi-square statistic. z^2 is distributed chi-square with 1 degree of freedom, though note that the user will likely need to turn off Yates' continuity correction by setting correct = FALSE to see this connection.
...	Additional arguments for prop.test().

Details

When testing with an explanatory variable with more than two levels, the order argument as used in the package is no longer well-defined. The function will thus raise a warning and ignore the value if supplied a non-NULL order argument.

The columns present in the output depend on the output of both prop.test() and broom::glance.htest(). See the latter's documentation for column definitions; columns have been renamed with the following mapping:

- chisq_df = parameter
- p_value = p.value
- lower_ci = conf.low
- upper_ci = conf.high

See Also

Other wrapper functions: chisq_stat(), chisq_test(), observe(), t_stat(), t_test()

Examples

```
# two-sample proportion test for difference in proportions of
# college completion by respondent sex
prop_test(gss,
          college ~ sex,
          order = c("female", "male"))

# one-sample proportion test for hypothesized null
# proportion of college completion of .2
prop_test(gss,
          college ~ NULL,
          p = .2)

# report as a z-statistic rather than chi-square
# and specify the success level of the response
prop_test(gss,
          college ~ NULL,
          success = "degree",
          p = .2,
          z = TRUE)
```

`rep_sample_n` *Perform repeated sampling*

Description

These functions extend the functionality of `dplyr::sample_n()` and `dplyr::slice_sample()` by allowing for repeated sampling of data. This operation is especially helpful while creating sampling distributions—see the examples below!

Usage

```
rep_sample_n(tbl, size, replace = FALSE, reps = 1, prob = NULL)

rep_slice_sample(
  .data,
  n = NULL,
  prop = NULL,
  replace = FALSE,
  weight_by = NULL,
  reps = 1
)
```

Arguments

`tbl, .data`	Data frame of population from which to sample.

`size, n, prop` `size` and `n` refer to the sample size of each sample. The `size` argument to `rep_sample_n()` is required, while in `rep_slice_sample()` sample size defaults to 1 if not specified. `prop`, an argument to `rep_slice_sample()`, refers to the proportion of rows to sample in each sample, and is rounded down in the case that `prop * nrow(.data)` is not an integer. When using `rep_slice_sample()`, please only supply one of `n` or `prop`.

`replace` Should samples be taken with replacement?

`reps` Number of samples to take.

`prob, weight_by` A vector of sampling weights for each of the rows in `.data`—must have length equal to `nrow(.data)`. For `weight_by`, this may also be an unquoted column name in `.data`.

Details

`rep_sample_n()` and `rep_slice_sample()` are designed to behave similar to their dplyr counterparts. As such, they have at least the following differences:

- In case `replace = FALSE` having `size` bigger than number of data rows in `rep_sample_n()` will give an error. In `rep_slice_sample()` having such `n` or `prop > 1` will give warning and output sample size will be set to number of rows in data.

Note that the `dplyr::sample_n()` function has been superseded by `dplyr::slice_sample()`.

Value

A tibble of size `reps * n` rows corresponding to `reps` samples of size `n` from `.data`, grouped by `replicate`.

Examples

```
library(dplyr)
library(ggplot2)
library(tibble)

# take 1000 samples of size n = 50, without replacement
slices <- gss %>%
  rep_slice_sample(n = 50, reps = 1000)

slices

# compute the proportion of respondents with a college
# degree in each replicate
p_hats <- slices %>%
  group_by(replicate) %>%
  summarize(prop_college = mean(college == "degree"))

# plot sampling distribution
ggplot(p_hats, aes(x = prop_college)) +
  geom_density() +
  labs(
```

```
    x = "p_hat", y = "Number of samples",
    title = "Sampling distribution of p_hat"
  )

# sampling with probability weights. Note probabilities are automatically
# renormalized to sum to 1
df <- tibble(
  id = 1:5,
  letter = factor(c("a", "b", "c", "d", "e"))
)

rep_slice_sample(df, n = 2, reps = 5, weight_by = c(.5, .4, .3, .2, .1))

# alternatively, pass an unquoted column name in `.data` as `weight_by`
df <- df %>% mutate(wts = c(.5, .4, .3, .2, .1))

rep_slice_sample(df, n = 2, reps = 5, weight_by = wts)
```

`shade_confidence_interval` *Add information about confidence interval*

Description

`shade_confidence_interval()` plots a confidence interval region on top of `visualize()` output. The output is a ggplot2 layer that can be added with `+`. The function has a shorter alias, `shade_ci()`.

Learn more in `vignette("infer")`.

Usage

```
shade_confidence_interval(
  endpoints,
  color = "mediumaquamarine",
  fill = "turquoise",
  ...
)

shade_ci(endpoints, color = "mediumaquamarine", fill = "turquoise", ...)
```

Arguments

`endpoints`	The lower and upper bounds of the interval to be plotted. Likely, this will be the output of `get_confidence_interval()`. For `calculate()`-based workflows, this will be a 2-element vector or a `1 x 2` data frame containing the lower and upper values to be plotted. For `fit()`-based workflows, a `(p + 1) x 3` data frame with columns `term`, `lower_ci`, and `upper_ci`, giving the upper and lower bounds for each regression term. For use in visualizations of `assume()` output, this must be the output of `get_confidence_interval()`.

color	A character or hex string specifying the color of the end points as a vertical lines on the plot.
fill	A character or hex string specifying the color to shade the confidence interval. If NULL then no shading is actually done.
...	Other arguments passed along to \ggplot2\ functions.

Value

If added to an existing infer visualization, a \ggplot2\ object displaying the supplied intervals on top of its corresponding distribution. Otherwise, an infer_layer list.

See Also

Other visualization functions: shade_p_value()

Examples

```
# find the point estimate---mean number of hours worked per week
point_estimate <- gss %>%
  specify(response = hours) %>%
  calculate(stat = "mean")

# ...and a bootstrap distribution
boot_dist <- gss %>%
  # ...we're interested in the number of hours worked per week
  specify(response = hours) %>%
  # generating data points
  generate(reps = 1000, type = "bootstrap") %>%
  # finding the distribution from the generated data
  calculate(stat = "mean")

# find a confidence interval around the point estimate
ci <- boot_dist %>%
  get_confidence_interval(point_estimate = point_estimate,
                          # at the 95% confidence level
                          level = .95,
                          # using the standard error method
                          type = "se")


# and plot it!
boot_dist %>%
  visualize() +
  shade_confidence_interval(ci)

# or just plot the bounds
boot_dist %>%
  visualize() +
  shade_confidence_interval(ci, fill = NULL)

# you can shade confidence intervals on top of
```

```
# theoretical distributions, too---the theoretical
# distribution will be recentered and rescaled to
# align with the confidence interval
sampling_dist <- gss %>%
  specify(response = hours) %>%
  assume(distribution = "t")

visualize(sampling_dist) +
  shade_confidence_interval(ci)


# to visualize distributions of coefficients for multiple
# explanatory variables, use a `fit()`-based workflow

# fit 1000 linear models with the `hours` variable permuted
null_fits <- gss %>%
 specify(hours ~ age + college) %>%
 hypothesize(null = "independence") %>%
 generate(reps = 1000, type = "permute") %>%
 fit()

null_fits

# fit a linear model to the observed data
obs_fit <- gss %>%
  specify(hours ~ age + college) %>%
  fit()

obs_fit

# get confidence intervals for each term
conf_ints <-
  get_confidence_interval(
    null_fits,
    point_estimate = obs_fit,
    level = .95
  )

# visualize distributions of coefficients
# generated under the null
visualize(null_fits)

# add a confidence interval shading layer to juxtapose
# the null fits with the observed fit for each term
visualize(null_fits) +
  shade_confidence_interval(conf_ints)


# more in-depth explanation of how to use the infer package
## Not run:
vignette("infer")

## End(Not run)
```

shade_p_value *Shade histogram area beyond an observed statistic*

Description

shade_p_value() plots a p-value region on top of visualize() output. The output is a ggplot2 layer that can be added with +. The function has a shorter alias, shade_pvalue().

Learn more in vignette("infer").

Usage

```
shade_p_value(obs_stat, direction, color = "red2", fill = "pink", ...)

shade_pvalue(obs_stat, direction, color = "red2", fill = "pink", ...)
```

Arguments

obs_stat	The observed statistic or estimate. For calculate()-based workflows, this will be a 1-element numeric vector or a 1 x 1 data frame containing the observed statistic. For fit()-based workflows, a (p + 1) x 2 data frame with columns term and estimate giving the observed estimate for each term.
direction	A string specifying in which direction the shading should occur. Options are "less", "greater", or "two-sided". Can also give "left", "right", "both", "two_sided", "two sided", or "two.sided". If NULL, the function will not shade any area.
color	A character or hex string specifying the color of the observed statistic as a vertical line on the plot.
fill	A character or hex string specifying the color to shade the p-value region. If NULL, the function will not shade any area.
...	Other arguments passed along to \ggplot2\ functions. For expert use only.

Value

If added to an existing infer visualization, a \ggplot2\ object displaying the supplied statistic on top of its corresponding distribution. Otherwise, an infer_layer list.

See Also

Other visualization functions: shade_confidence_interval()

Examples

```
# find the point estimate---mean number of hours worked per week
point_estimate <- gss %>%
  specify(response = hours) %>%
  hypothesize(null = "point", mu = 40) %>%
  calculate(stat = "t")

# ...and a null distribution
null_dist <- gss %>%
  # ...we're interested in the number of hours worked per week
  specify(response = hours) %>%
  # hypothesizing that the mean is 40
  hypothesize(null = "point", mu = 40) %>%
  # generating data points for a null distribution
  generate(reps = 1000, type = "bootstrap") %>%
  # estimating the null distribution
  calculate(stat = "t")

# shade the p-value of the point estimate
null_dist %>%
  visualize() +
  shade_p_value(obs_stat = point_estimate, direction = "two-sided")

# you can shade confidence intervals on top of
# theoretical distributions, too!
null_dist_theory <- gss %>%
  specify(response = hours) %>%
  assume(distribution = "t")

null_dist_theory %>%
  visualize() +
  shade_p_value(obs_stat = point_estimate, direction = "two-sided")


# to visualize distributions of coefficients for multiple
# explanatory variables, use a `fit()`-based workflow

# fit 1000 linear models with the `hours` variable permuted
null_fits <- gss %>%
 specify(hours ~ age + college) %>%
 hypothesize(null = "independence") %>%
 generate(reps = 1000, type = "permute") %>%
 fit()

null_fits

# fit a linear model to the observed data
obs_fit <- gss %>%
  specify(hours ~ age + college) %>%
  fit()

obs_fit
```

```
# visualize distributions of coefficients
# generated under the null
visualize(null_fits)

# add a p-value shading layer to juxtapose the null
# fits with the observed fit for each term
visualize(null_fits) +
  shade_p_value(obs_fit, direction = "both")

# the direction argument will be applied
# to the plot for each term
visualize(null_fits) +
  shade_p_value(obs_fit, direction = "left")


# more in-depth explanation of how to use the infer package
## Not run:
vignette("infer")

## End(Not run)
```

`specify` *Specify response and explanatory variables*

Description

`specify()` is used to specify which columns in the supplied data frame are the relevant response (and, if applicable, explanatory) variables. Note that character variables are converted to `factors`.

Learn more in `vignette("infer")`.

Usage

```
specify(x, formula, response = NULL, explanatory = NULL, success = NULL)
```

Arguments

`x`	A data frame that can be coerced into a tibble.
`formula`	A formula with the response variable on the left and the explanatory on the right. Alternatively, a `response` and `explanatory` argument can be supplied.
`response`	The variable name in `x` that will serve as the response. This is an alternative to using the `formula` argument.
`explanatory`	The variable name in `x` that will serve as the explanatory variable. This is an alternative to using the formula argument.
`success`	The level of `response` that will be considered a success, as a string. Needed for inference on one proportion, a difference in proportions, and corresponding z stats.

Value

A tibble containing the response (and explanatory, if specified) variable data.

See Also

Other core functions: `calculate()`, `generate()`, `hypothesize()`

Examples

```
# specifying for a point estimate on one variable
gss %>%
   specify(response = age)

# specify a relationship between variables as a formula...
gss %>%
  specify(age ~ partyid)

# ...or with named arguments!
gss %>%
  specify(response = age, explanatory = partyid)

# more in-depth explanation of how to use the infer package
## Not run:
vignette("infer")

## End(Not run)
```

---

t_stat *Tidy t-test statistic*

Description

A shortcut wrapper function to get the observed test statistic for a t test. This function has been deprecated in favor of the more general `observe()`.

Usage

```
t_stat(
  x,
  formula,
  response = NULL,
  explanatory = NULL,
  order = NULL,
  alternative = "two-sided",
  mu = 0,
  conf_int = FALSE,
  conf_level = 0.95,
  ...
)
```

Arguments

x	A data frame that can be coerced into a tibble.
formula	A formula with the response variable on the left and the explanatory on the right.
response	The variable name in x that will serve as the response. This is alternative to using the formula argument.
explanatory	The variable name in x that will serve as the explanatory variable.
order	A string vector of specifying the order in which the levels of the explanatory variable should be ordered for subtraction, where order = c("first", "second") means ("first" - "second").
alternative	Character string giving the direction of the alternative hypothesis. Options are "two-sided" (default), "greater", or "less".
mu	A numeric value giving the hypothesized null mean value for a one sample test and the hypothesized difference for a two sample test.
conf_int	A logical value for whether to include the confidence interval or not. TRUE by default.
conf_level	A numeric value between 0 and 1. Default value is 0.95.
...	Pass in arguments to \infer\ functions.

See Also

Other wrapper functions: chisq_stat(), chisq_test(), observe(), prop_test(), t_test()

Other functions for calculating observed statistics: chisq_stat(), observe()

Examples

```
library(tidyr)

# t test statistic for true mean number of hours worked
# per week of 40
gss %>%
   t_stat(response = hours, mu = 40)

# t test statistic for number of hours worked per week
# by college degree status
gss %>%
   tidyr::drop_na(college) %>%
   t_stat(formula = hours ~ college,
      order = c("degree", "no degree"),
      alternative = "two-sided")
```

`t_test` *Tidy t-test*

Description

A tidier version of t.test() for two sample tests.

Usage

```
t_test(
  x,
  formula,
  response = NULL,
  explanatory = NULL,
  order = NULL,
  alternative = "two-sided",
  mu = 0,
  conf_int = TRUE,
  conf_level = 0.95,
  ...
)
```

Arguments

`x`	A data frame that can be coerced into a tibble.
`formula`	A formula with the response variable on the left and the explanatory on the right.
`response`	The variable name in `x` that will serve as the response. This is alternative to using the `formula` argument.
`explanatory`	The variable name in `x` that will serve as the explanatory variable.
`order`	A string vector of specifying the order in which the levels of the explanatory variable should be ordered for subtraction, where `order = c("first", "second")` means `("first" - "second")`.
`alternative`	Character string giving the direction of the alternative hypothesis. Options are `"two-sided"` (default), `"greater"`, or `"less"`.
`mu`	A numeric value giving the hypothesized null mean value for a one sample test and the hypothesized difference for a two sample test.
`conf_int`	A logical value for whether to include the confidence interval or not. `TRUE` by default.
`conf_level`	A numeric value between 0 and 1. Default value is 0.95.
`...`	For passing in other arguments to t.test().

See Also

Other wrapper functions: `chisq_stat()`, `chisq_test()`, `observe()`, `prop_test()`, `t_stat()`

Examples

```
library(tidyr)

# t test for number of hours worked per week
# by college degree status
gss %>%
   tidyr::drop_na(college) %>%
   t_test(formula = hours ~ college,
      order = c("degree", "no degree"),
      alternative = "two-sided")

# see vignette("infer") for more explanation of the
# intuition behind the infer package, and vignette("t_test")
# for more examples of t-tests using infer
```

`visualize` *Visualize statistical inference*

Description

Visualize the distribution of the simulation-based inferential statistics or the theoretical distribution (or both!).

Learn more in `vignette("infer")`.

Usage

```
visualize(data, bins = 15, method = "simulation", dens_color = "black", ...)

visualise(data, bins = 15, method = "simulation", dens_color = "black", ...)
```

Arguments

`data`	A distribution. For simulation-based inference, a data frame containing a distribution of `calculate()`d statistics or `fit()`ted coefficient estimates. This object should have been passed to `generate()` before being supplied or `calculate()` to `fit()`. For theory-based inference, the output of `assume()`.
`bins`	The number of bins in the histogram.
`method`	A string giving the method to display. Options are `"simulation"`, `"theoretical"`, or `"both"` with `"both"` corresponding to `"simulation"` and `"theoretical"`. If `data` is the output of `assume()`, this argument will be ignored and default to `"theoretical"`.
`dens_color`	A character or hex string specifying the color of the theoretical density curve.
`...`	Other arguments passed along to \ggplot2\ functions.

Details

In order to make the visualization workflow more straightforward and explicit, `visualize()` now only should be used to plot distributions of statistics directly. A number of arguments related to shading p-values and confidence intervals are now deprecated in `visualize()` and should now be passed to `shade_p_value()` and `shade_confidence_interval()`, respectively. `visualize()` will raise a warning if deprecated arguments are supplied.

Value

For `calculate()`-based workflows, a ggplot showing the simulation-based distribution as a histogram or bar graph. Can also be used to display theoretical distributions.

For `assume()`-based workflows, a ggplot showing the theoretical distribution.

For `fit()`-based workflows, a `patchwork` object showing the simulation-based distributions as a histogram or bar graph. The interface to adjust plot options and themes is a bit different for `patchwork` plots than ggplot2 plots. The examples highlight the biggest differences here, but see `patchwork::plot_annotation()` and patchwork::&.gg for more details.

See Also

`shade_p_value()`, `shade_confidence_interval()`.

Examples

```
# generate a null distribution
null_dist <- gss %>%
  # we're interested in the number of hours worked per week
  specify(response = hours) %>%
  # hypothesizing that the mean is 40
  hypothesize(null = "point", mu = 40) %>%
  # generating data points for a null distribution
  generate(reps = 1000, type = "bootstrap") %>%
  # calculating a distribution of means
  calculate(stat = "mean")

# or a bootstrap distribution, omitting the hypothesize() step,
# for use in confidence intervals
boot_dist <- gss %>%
  specify(response = hours) %>%
  generate(reps = 1000, type = "bootstrap") %>%
  calculate(stat = "mean")

# we can easily plot the null distribution by piping into visualize
null_dist %>%
  visualize()

# we can add layers to the plot as in ggplot, as well...
# find the point estimate---mean number of hours worked per week
point_estimate <- gss %>%
  specify(response = hours) %>%
```

```
  calculate(stat = "mean")

# find a confidence interval around the point estimate
ci <- boot_dist %>%
  get_confidence_interval(point_estimate = point_estimate,
                          # at the 95% confidence level
                          level = .95,
                          # using the standard error method
                          type = "se")

# display a shading of the area beyond the p-value on the plot
null_dist %>%
  visualize() +
  shade_p_value(obs_stat = point_estimate, direction = "two-sided")

# ...or within the bounds of the confidence interval
null_dist %>%
  visualize() +
  shade_confidence_interval(ci)

# plot a theoretical sampling distribution by creating
# a theory-based distribution with `assume()`
sampling_dist <- gss %>%
  specify(response = hours) %>%
  assume(distribution = "t")

visualize(sampling_dist)

# you can shade confidence intervals on top of
# theoretical distributions, too---the theoretical
# distribution will be recentered and rescaled to
# align with the confidence interval
visualize(sampling_dist) +
  shade_confidence_interval(ci)


# to plot both a theory-based and simulation-based null distribution,
# use a theorized statistic (i.e. one of t, z, F, or Chisq)
# and supply the simulation-based null distribution
null_dist_t <- gss %>%
  specify(response = hours) %>%
  hypothesize(null = "point", mu = 40) %>%
  generate(reps = 1000, type = "bootstrap") %>%
  calculate(stat = "t")

obs_stat <- gss %>%
  specify(response = hours) %>%
  hypothesize(null = "point", mu = 40) %>%
  calculate(stat = "t")

visualize(null_dist_t, method = "both")

visualize(null_dist_t, method = "both") +
```

```
  shade_p_value(obs_stat, "both")


# to visualize distributions of coefficients for multiple
# explanatory variables, use a `fit()`-based workflow

# fit 1000 models with the `hours` variable permuted
null_fits <- gss %>%
 specify(hours ~ age + college) %>%
 hypothesize(null = "independence") %>%
 generate(reps = 1000, type = "permute") %>%
 fit()

null_fits

# visualize distributions of resulting coefficients
visualize(null_fits)

# the interface to add themes and other elements to patchwork
# plots (outputted by `visualize` when the inputted data
# is from the `fit()` function) is a bit different than adding
# them to ggplot2 plots.
library(ggplot2)

# to add a ggplot2 theme to a `calculate()`-based visualization, use `+`
null_dist %>% visualize() + theme_dark()

# to add a ggplot2 theme to a `fit()`-based visualization, use `&`
null_fits %>% visualize() & theme_dark()


# More in-depth explanation of how to use the infer package
## Not run:
vignette("infer")

## End(Not run)
```

`%>%` *Pipe*

Description

Like {dplyr}, {infer} also uses the pipe (`%>%`) function from `magrittr` to turn function composition into a series of iterative statements.

Arguments

`lhs, rhs`	Inference functions and the initial data frame.

Index