ggstatsplot: ggplot2 Based Plots with Statistical Details

Indrajeet Patil* Mina Cikara†

2020-01-21

Contents

*Harvard University, patilindrajeet.science@gmail.com
†Harvard University

> "What is to be sought in designs for the display of information is the clear portrayal of complexity. Not the complication of the simple; rather ... the revelation of the complex."
> - Edward R. Tufte

1 Introduction: Raison d'être

ggstatsplot is an extension of ggplot2 package for creating graphics with details from statistical tests included in the plots themselves and targeted primarily at behavioral sciences community to provide a one-line code to produce information-rich plots. In a typical exploratory data analysis workflow, data visualization and statistical modeling are two different phases: visualization informs modeling, and modeling in its turn can suggest a different visualization method, and so on and so forth. The central idea of ggstatsplot is simple: combine these two phases into one in the form of graphics with statistical details, which makes data exploration simpler and faster.

1.1 Need for informative visualizations

1.2 Need for better statistical reporting

But why would combining statistical analysis with data visualization be helpful? We list few reasons below-

- A recent survey (Nuijten, Hartgerink, van Assen, Epskamp, & Wicherts, 2016) revealed that one in eight papers in major psychology journals contained a grossly inconsistent p-value that may have affected the statistical conclusion. ggstatsplot helps avoid such reporting errors: Since the plot and the statistical analysis are yoked together, the chances of making an error in reporting the results are minimized. One need not write the results manually or copy-paste them from a different statistics software program (like SPSS, SAS, and so on).

2 ggstatsplot at a glance

2.1 Summary of types of plots included

It produces a limited kinds of ready-made plots for the supported analyses:

Function	Plot	Description
`ggbetweenstats`	**violin plots**	for comparisons *between* groups/conditions
`ggwithinstats`	**violin plots**	for comparisons *within* groups/conditions
`gghistostats`	**histograms**	for distribution about numeric variable
`ggdotplotstats`	**dot plots/charts**	for distribution about labeled numeric variable
`ggpiestats`	**pie charts**	for categorical data
`ggbarstats`	**bar charts**	for categorical data
`ggscatterstats`	**scatterplots**	for correlations between two variables
`ggcorrmat`	**correlation matrices**	for correlations between multiple variables
`ggcoefstats`	**dot-and-whisker plots**	for regression models

In addition to these basic plots, `ggstatsplot` also provides `grouped_` versions (see below) that makes it easy to repeat the same analysis for any grouping variable.

2.2 Summary of types of statistical analyses

Most functions share a `type` (of test) argument that is helpful to specify the type of statistical analysis:

- `"parametric"` (for **parametric** tests)
- `"nonparametric"` (for **non-parametric** tests)
- `"robust"` (for **robust** tests)
- `"bayes"` (for **Bayes Factor** tests)

The table below summarizes all the different types of analyses currently supported in this package-

Functions	Description	Parametric	Non-parametric	Robust	Bayes Factor
`ggbetweenstats`	Between group/condition comparisons	Yes	Yes	Yes	Yes
`ggwithinstats`	Within group/condition comparisons	Yes	Yes	Yes	Yes
`gghistostats`, `ggdotplotstats`	Distribution of a numeric variable	Yes	Yes	Yes	Yes
`ggcorrmat`	Correlation matrix	Yes	Yes	Yes	No
`ggscatterstats`	Correlation between two variables	Yes	Yes	Yes	Yes
`ggpiestats`, `ggbarstats`	Association between categorical variables	Yes	`NA`	`NA`	Yes
`ggpiestats`, `ggbarstats`	Equal proportions for categorical variable levels	Yes	`NA`	`NA`	Yes
`ggcoefstats`	Regression model coefficients	Yes	No	Yes	No
`ggcoefstats`	Random-effects meta-analysis	Yes	No	No	Yes

In the following sections, we will discuss at depth justification for why the plots have been designed in certain ways and what principles were followed to report statistical details on the plots.

3 Graphic design principles

3.1 Graphical perception

Graphical perception involves visual decoding of the encoded information in graphs. `ggstatsplot` incorporates the paradigm proposed in ((Cleveland, 1985), Chapter 4) to facilitate making visual judgments about quantitative information effortless and almost instantaneous. Based on experiments, Cleveland proposes that there are ten elementary graphical-perception tasks that we perform to visually decode quantitative information in graphs (organized from most to least accurate; (Cleveland, 1985), p.254)-

- Position along a common scale
- Position along identical, non-aligned scales
- Length
- Angle (Slope)
- Area
- Volume
- Color hue

So the key principle of Cleveland's paradigm for data display is-

> "We should encode data on a graph so that the visual decoding involves [graphical-perception] tasks as high in the ordering as possible."

For example, decoding the data point values in `ggbetweenstats` requires position judgments along a common scale (Figure 1):

```
# for reproducibility
set.seed(123)

# plot
ggstatsplot::ggbetweenstats(
  data = dplyr::filter(
    .data = ggstatsplot::movies_long,
    genre %in% c("Action", "Action Comedy", "Action Drama", "Comedy")
  ),
  x = genre,
  y = rating,
  title = "IMDB rating by film genre",
  xlab = "Genre",
  ylab = "IMDB rating (average)",
  pairwise.comparisons = TRUE,
  p.adjust.method = "bonferroni",
  ggtheme = hrbrthemes::theme_ipsum_tw(),
  ggstatsplot.layer = FALSE,
  outlier.tagging = TRUE,
  outlier.label = title,
  messages = FALSE
)
```

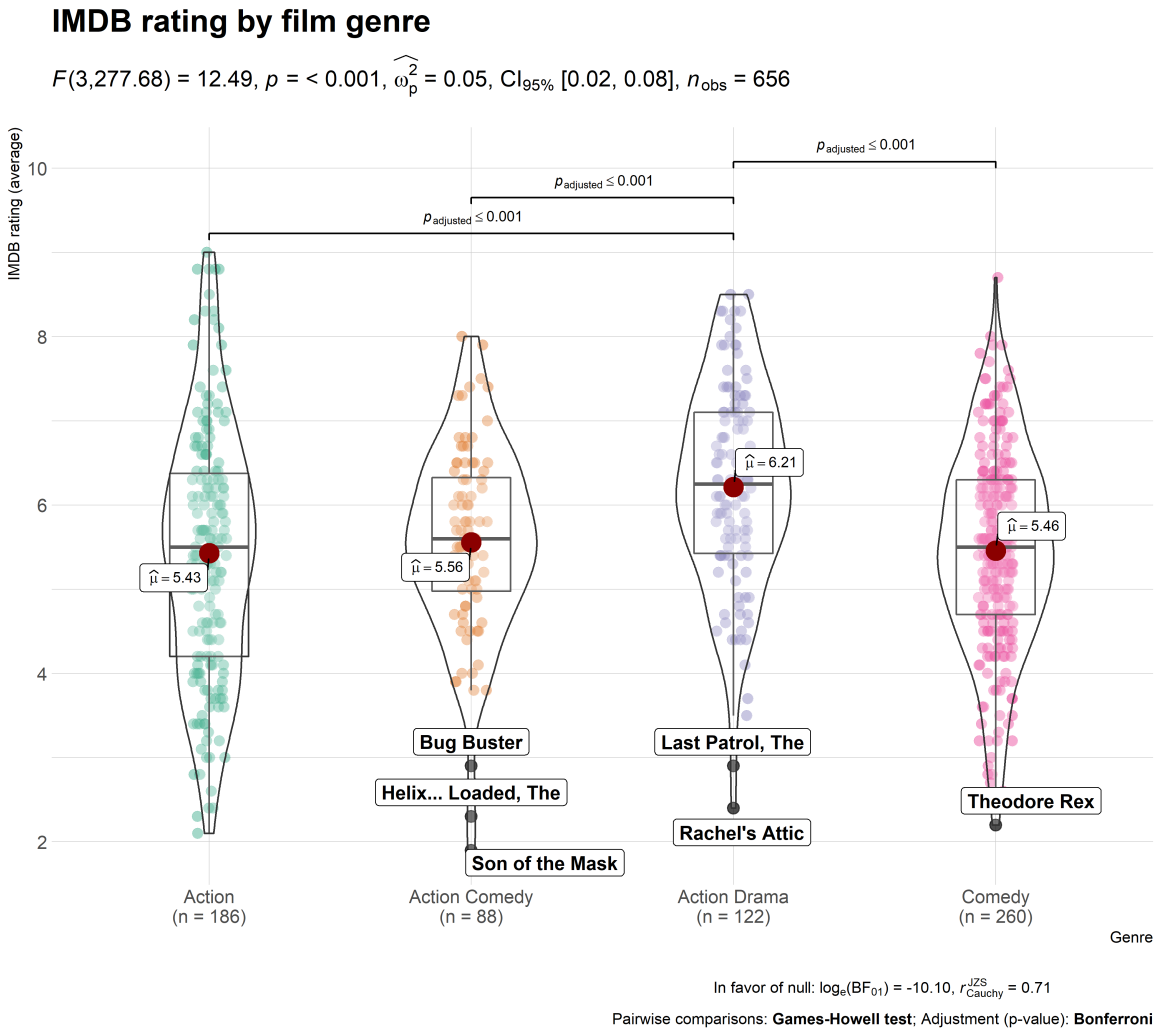


Figure 1: Note that assessing differences in mean values between groups has been made easier with the help of *position* of data points along a common scale (the Y-axis) and labels.

There are few instances where `ggstatsplot` diverges from recommendations made in Cleveland's paradigm:

- For the categorical/nominal data, `ggstatsplot` uses pie charts (see Figure 2) which rely on *angle* judgments, which are less accurate (as compared to bar graphs, e.g., which require *position* judgments). This shortcoming is assuaged to some degree by using plenty of labels that describe percentages for all slices. This makes angle judgment unnecessary and pre-vacates any concerns about inaccurate judgments about percentages. Additionally, it also provides alternative function to `ggpiestats` for working with categorical variables: `ggbarstats`.

```
# for reproducibility
set.seed(123)

# plot
```

```
ggstatsplot::ggpiestats(
  data = ggstatsplot::movies_long,
  x = genre,
  y = mpaa,
  title = "Distribution of MPAA ratings by film genre",
  legend.title = "layout",
  caption = substitute(paste(
    italic("MPAA"), ": Motion Picture Association of America"
  )),
  palette = "Paired",
  messages = FALSE
)
```

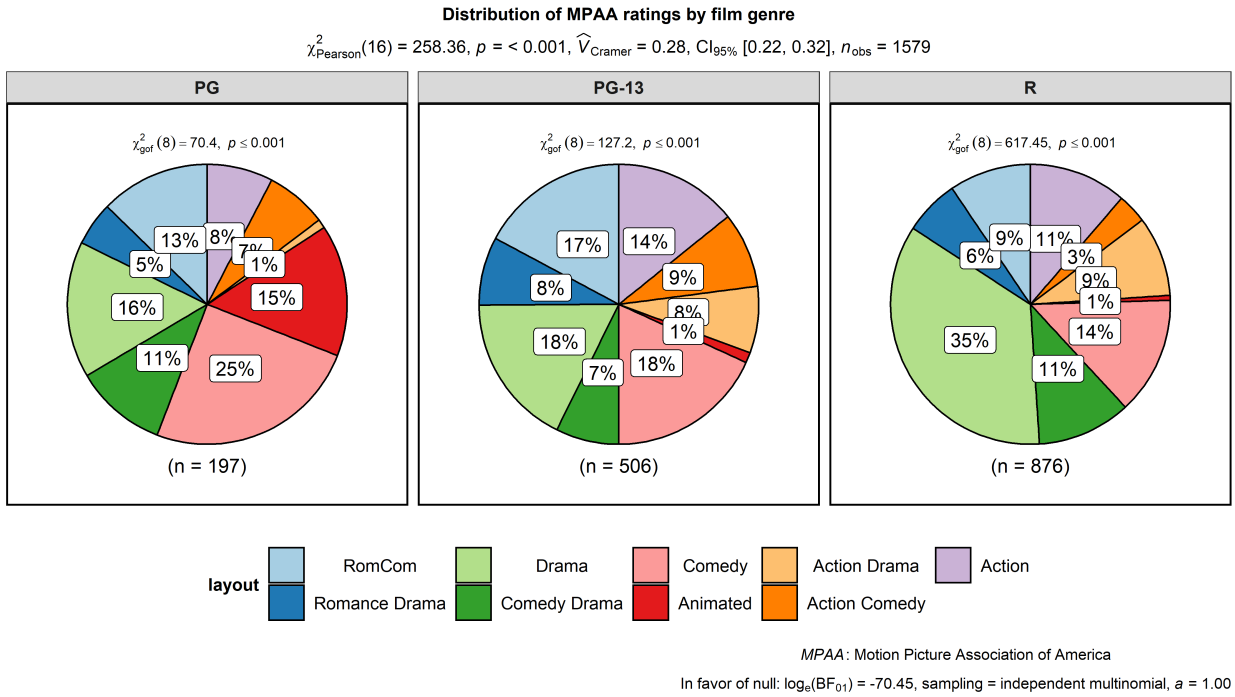


Figure 2: Pie charts don't follow Cleveland's paradigm to data display because they rely on less accurate angle judgments. 'ggstatsplot' sidesteps this issue by always labelling percentages for pie slices, which makes angle judgments unnecessary.

- Cleveland's paradigm also emphasizes that *superposition* of data is better than *juxtaposition* ((Cleveland, 1985), p.201) because this allows for a more incisive comparison of the values from different parts of the dataset. This recommendation is violated in all `grouped_` variants of the function (see Figure 3). Note that the range for Y-axes are no longer the same across juxtaposed subplots and so visually comparing the data becomes difficult. On the other hand, in the superposed plot, all data have the same range and coloring different parts makes the visual discrimination of different components of the data, and their comparison, easier. But the goal of `grouped_` variants of functions is to not only show different aspects of the data but also to run statistical tests and showing detailed results for all aspects of the data in a superposed plot is difficult. Therefore, this is a compromise `ggstatsplot` is comfortable with, at least to produce plots for quick exploration of different aspects of the data.

```r
# for reproducibility
set.seed(123)
library(ggplot2)

# creating a smaller dataframe
df <- dplyr::filter(ggstatsplot::movies_long, genre %in% c("Comedy", "Drama"))

# plot
ggstatsplot::combine_plots(
  # plot 1: superposition
  ggplot(data = df, mapping = ggplot2::aes(x = length, y = rating, color = genre)) +
    geom_jitter(size = 3, alpha = 0.5) +
    geom_smooth(method = "lm") +
    labs(title = "superposition (recommended in Cleveland's paradigm)") +
    ggstatsplot::theme_ggstatsplot(),
  # plot 2: juxtaposition
  ggstatsplot::grouped_ggscatterstats(
    data = df,
    x = length,
    y = rating,
    grouping.var = genre,
    marginal = FALSE,
    messages = FALSE,
    title.prefix = "Genre",
    title.text = "juxtaposition (`ggstatsplot` implementation in `grouped_` functions)",
    title.size = 12
  ),
  # combine for comparison
  title.text = "Two ways to compare different aspects of data",
  nrow = 2,
  labels = c("(a)", "(b)")
)
```

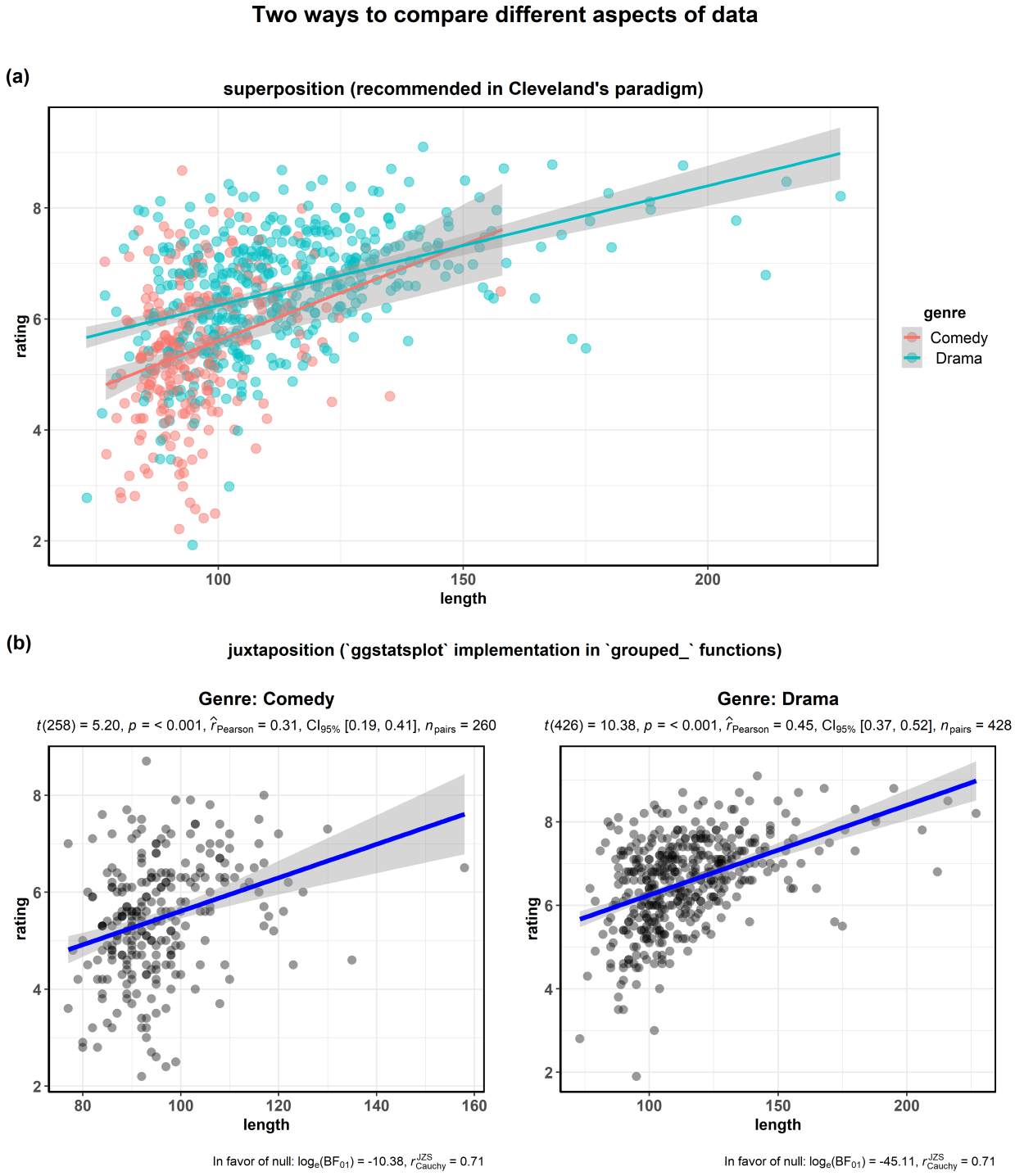


Figure 3: Comparing different aspects of data is much more accurate in (*a*) a *superposed* plot, which is recommended in Cleveland's paradigm, than in (*b*) a *juxtaposed* plot, which is how it is implemented in 'ggstatsplot' package. This is because displaying detailed results from statistical tests would be difficult in a superposed plot.

The `grouped_` plots follow the *Shrink Principle* ((Tufte, 2001), p.166-7) for high-information graphics, which dictates that the data density and the size of the data matrix can be maximized to exploit maximum resolution of the available data-display technology. Given the large maximum resolution afforded by most computer monitors today, saving `grouped_` plots with appropriate resolution ensures no loss in legibility with reduced graphics area.

3.2 Graphical excellence

Graphical excellence consists of communicating complex ideas with clarity and in a way that the viewer understands the greatest number of ideas in a short amount of time all the while not quoting the data out of context. The package follows the principles for *graphical integrity* (Tufte, 2001):

- The physical representation of numbers is proportional to the numerical quantities they represent (e.g., Figure 1 and Figure 2 show how means (in `ggbetweenstats`) or percentages (`ggpiestats`) are proportional to the vertical distance or the area, respectively).
- All important events in the data have clear, detailed, and thorough labeling (e.g., Figure 1 plot shows how `ggbetweenstats` labels means, sample size information, outliers, and pairwise comparisons; same can be appreciated for `ggpiestats` in Figure 2 and `gghistostats` in Figure 4). Note that data labels in the data region are designed in a way that they don't interfere with our ability to assess the overall pattern of the data ((Cleveland, 1985); p.44-45). This is achieved by using `ggrepel` package to place labels in a way that reduces their visual prominence.
- None of the plots have *design* variation (e.g., abrupt change in scales) over the surface of a same graphic because this can lead to a false impression about variation in *data*.
- The number of information-carrying dimensions never exceed the number of dimensions in the data (e.g., using area to show one-dimensional data).
- All plots are designed to have no **chartjunk** (like moiré vibrations, fake perspective, dark grid lines, etc.) ((Tufte, 2001), Chapter 5).

There are some instances where `ggstatsplot` graphs don't follow principles of clean graphics, as formulated in the Tufte theory of data graphics ((Tufte, 2001), Chapter 4). The theory has four key principles:

1. Above all else show the data.
2. Maximize the data-ink ratio.
3. Erase non-data-ink.
4. Erase redundant data-ink, within reason.

In particular, default plots in `ggstatsplot` can sometimes violate one of the principles from 2-4. According to these principles, every bit of ink should have reason for its inclusion in the graphic and should convey some new information to the viewer. If not, such ink should be removed. One instance of this is bilateral symmetry of data measures. For example, in Figure 1, we can see that both the box and violin plots are mirrored, which consumes twice the space in the graphic without adding any new information. But this redundancy is tolerated for the sake of beauty that such symmetrical shapes can bring to the graphic. Even Tufte admits that efficiency is but one consideration in the design of statistical graphics ((Tufte, 2001), p. 137). Additionally, these principles were formulated in an era in which computer graphics had yet to revolutionize the ease with which graphics could be produced and thus some of the concerns about minimizing data-ink for easier production of graphics are not as relevant as they were.

3.3 Statistical variation

One of the important functions of a plot is to show the variation in the data, which comes in two forms:

- **Measurement noise**: In `ggstatsplot`, the actual variation in measurements is shown by plotting a combination of (jittered) raw data points with a boxplot laid on top (Figure 1) or a histogram (Figure 4). None of the plots, where empirical distribution of the data is concerned, show the sample standard deviation because they are poor at conveying information about limits of the sample and presence of outliers ((Cleveland, 1985), p.220).

```
# for reproducibility
set.seed(123)

# plot
ggstatsplot::gghistostats(
  data = morley,
  x = Speed,
  test.value = 792,
  test.value.line = TRUE,
  xlab = "Speed of light (km/sec, with 299000 subtracted)",
  title = "Distribution of measured Speed of light",
  caption = "Note: Data collected across 5 experiments (20 measurements each)",
  messages = FALSE
)
```

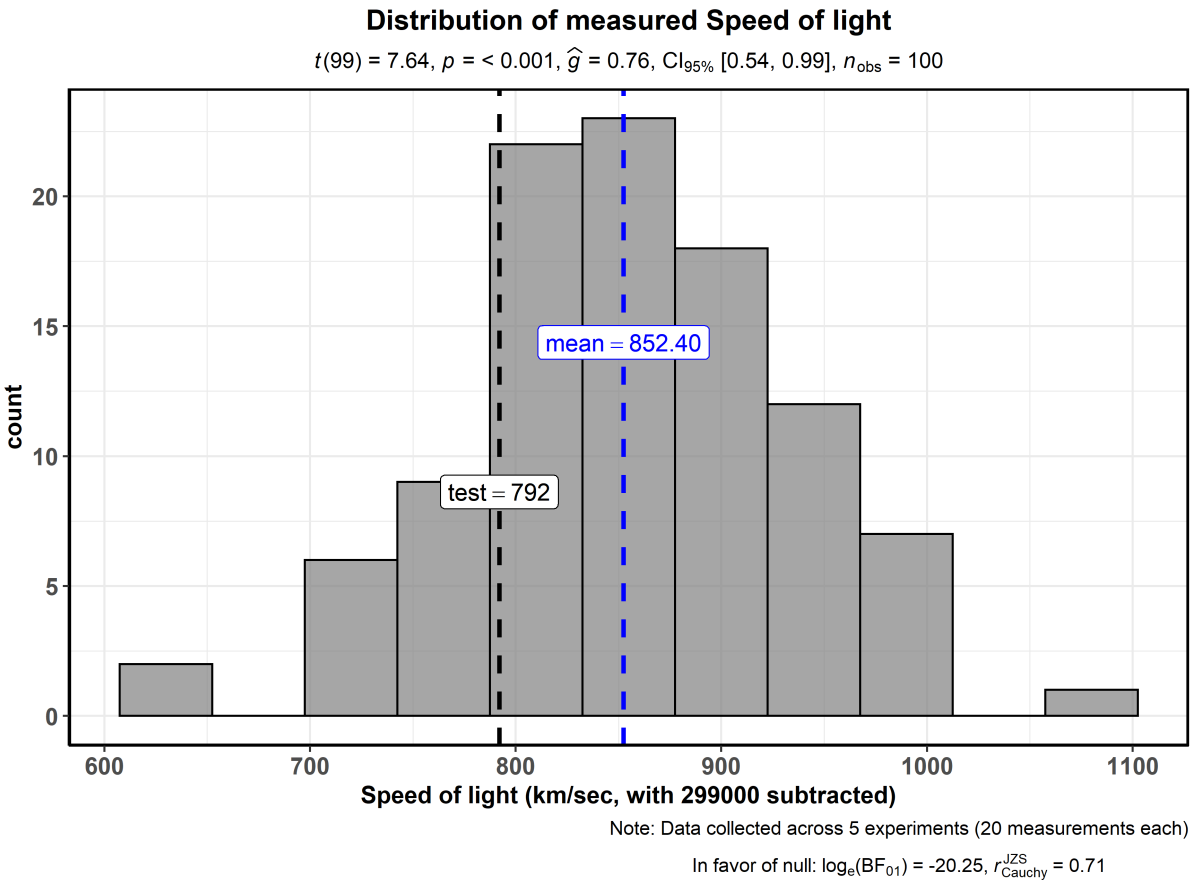


Figure 4: Distribution of a variable shown using 'gghistostats'.

- **Sample-to-sample statistic variation**: Although, traditionally, this variation has been shown using the standard error of the mean (SEM) of the statistic, `ggstatsplot` plots instead use 95% confidence intervals (e.g., Figure 5). This is because the interval formed by error bars correspond to a 68% confidence interval, which is not a particularly interesting interval ((Cleveland, 1985), p.222-225).

```
# for reproducibility
set.seed(123)

# creating model object
mod <- lme4::lmer(
  formula = total.fruits ~ nutrient + rack + (nutrient | gen),
  data = lme4::Arabidopsis
)

# plot
ggstatsplot::ggcoefstats(x = mod)
```

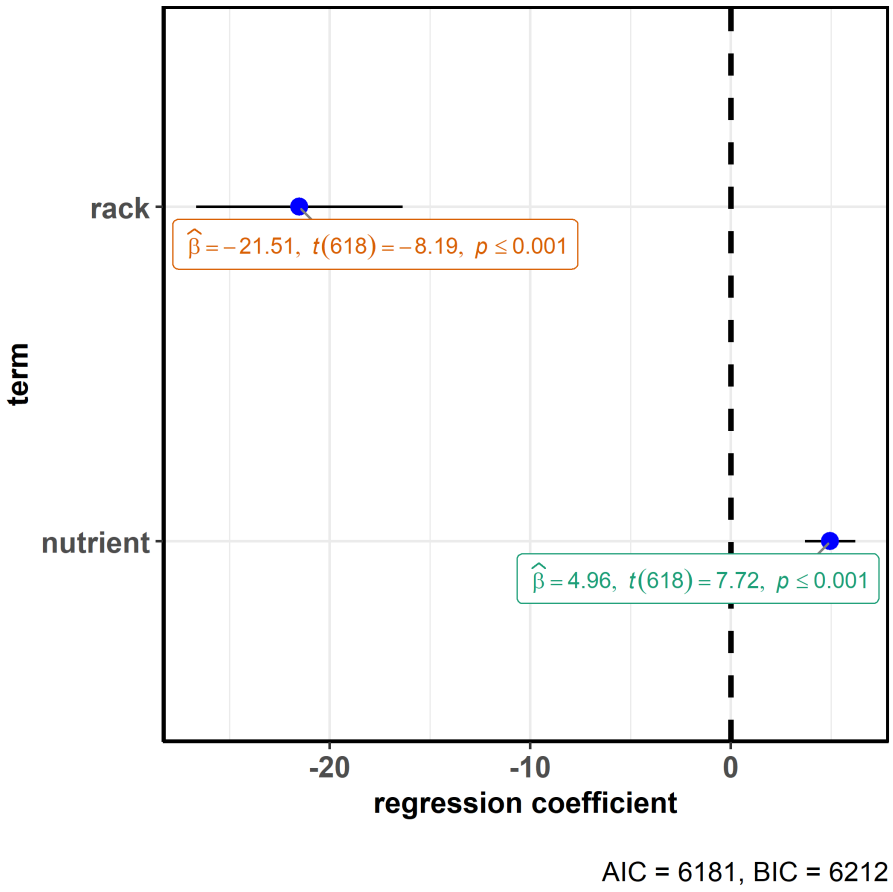


Figure 5: Sample-to-sample variation in regression estimates is displayed using confidence intervals in 'ggcoefstats'.

4 Statistical analysis

4.1 Data requirements

As an extension of `ggplot2`, `ggstatsplot` has the same expectations about the structure of the data. More specifically,

- The data should be organized following the principles of *tidy data*, which specify how statistical structure of a data frame (variables and observations) should be mapped to physical structure (columns and rows). More specifically, tidy data means all variables have their own columns and each row corresponds to a unique observation ((Wickham, 2014)).
- All `ggstatsplot` functions remove `NA`s from variables of interest (similar to `ggplot2`; (Wickham, 2016), p.207) in the data and display total sample size (n, either observations for between-subjects or pairs for within-subjects designs) in the subtitle to inform the user/reader about the number of observations included for both the statistical analysis and the visualization. But, when sample sizes differ *across* tests in the same function, `ggstatsplot` makes an effort to inform the user of this aspect. For example, `ggcorrmat` features several correlation test pairs and, depending on variables in a given pair, the sample sizes may vary (Figure 6).

```r
# for reproducibility
set.seed(123)

# creating a new dataset without any NAs in variables of interest
msleep_no_na <-
  dplyr::filter(
    .data = ggplot2::msleep,
    !is.na(sleep_rem), !is.na(awake), !is.na(brainwt), !is.na(bodywt)
  )

# variable names vector
var_names <- c("REM sleep", "time awake", "brain weight", "body weight")

# combining two plots using helper function in `ggstatsplot`
ggstatsplot::combine_plots(
  plotlist = purrr::pmap(
    .l = list(data = list(msleep_no_na, ggplot2::msleep)),
    .f = ggstatsplot::ggcorrmat,
    p.adjust.method = "holm",
    cor.vars = c(sleep_rem, awake:bodywt),
    cor.vars.names = var_names,
    matrix.type = "upper",
    colors = c("#B2182B", "white", "#4D4D4D"),
    title = "Correlalogram for mammals sleep dataset",
    subtitle = "sleep units: hours; weight units: kilograms",
    messages = FALSE
  ),
  labels = c("(a)", "(b)"),
  nrow = 1
)
```

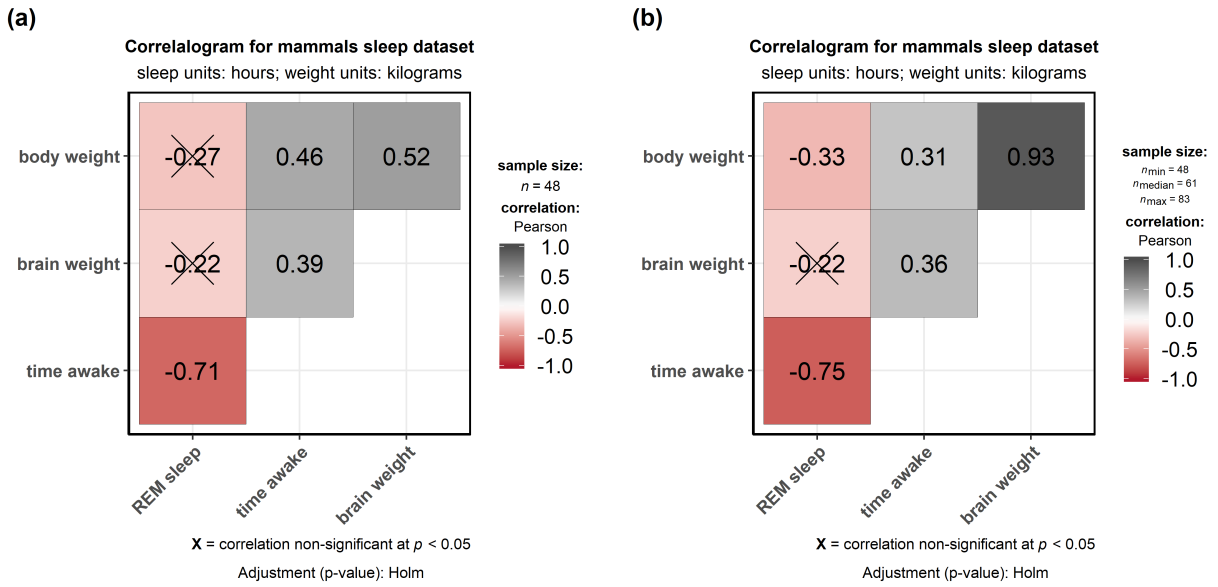


Figure 6: 'ggstatsplot' functions remove 'NA's from variables of interest and display total sample size n, but they can give more nuanced information about sample sizes when n differs across tests. For example, 'ggcorrmat' will display (a) only one total sample size once when no 'NA's present, but (b) will instead show minimum, median, and maximum sample sizes across all correlation tests when 'NA's are present across correlation variables.

4.2 Statistical reporting

The default setting in `ggstatsplot` is to produce plots with statistical details included. Most often than not, these results are displayed as a `subtitle` in the plot. Great care has been taken into which details are included in statistical reporting and why. For example, the template below is used to show results from Yuen's test for trimmed means (robust t-test):

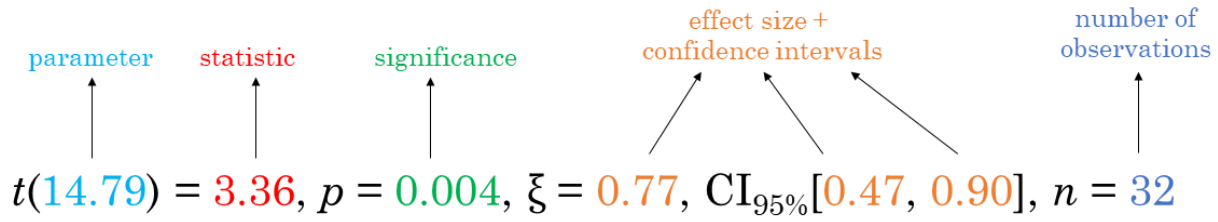


Figure 7: Template for reporting statistical details

APA guidelines (Association, 2009) are followed by default while reporting statistical details:

- Percentages are displayed with no decimal places (Figure 2).
- Correlations, t-tests, and χ^2-tests are reported with the degrees of freedom in parentheses and the significance level (Figure 6, Figure 3, Figure 4).
- ANOVAs are reported with two degrees of freedom and the significance level (Figure 1).

- Regression results are presented with the unstandardized or standardized estimate (beta), whichever was specified by the user, along with the statistic (depending on the model, this can be a t, F, or z statistic) and the corresponding significance level (Figure 5).
- With the exception of p-values, most statistics are rounded to two decimal places by default.

4.3 Statistical tests:

Here is a summary table of all the statistical tests currently supported across various functions:

Functions	Type	Test	Effect size	95% CI available?
`ggbetweenstats` (2 groups)	Parametric	Student's and Welch's t-test	Cohen's d, Hedge's g	✓
`ggbetweenstats` (> 2 groups)	Parametric	Fisher's and Welch's one-way ANOVA	$\eta^2, \eta_p^2, \omega^2, \omega_p^2$	✓
`ggbetweenstats` (2 groups)	Non-parametric	Mann-Whitney U-test	r	✓
`ggbetweenstats` (> 2 groups)	Non-parametric	Kruskal-Wallis Rank Sum Test	ϵ^2	✓
`ggbetweenstats` (2 groups)	Robust	Yuen's test for trimmed means	ξ	✓
`ggbetweenstats` (> 2 groups)	Robust	Heteroscedastic one-way ANOVA for trimmed means	ξ	✓
`ggwithinstats` (2 groups)	Parametric	Student's t-test	Cohen's d, Hedge's g	✓
`ggwithinstats` (> 2 groups)	Parametric	Fisher's one-way repeated measures ANOVA	η_p^2, ω^2	✓
`ggwithinstats` (2 groups)	Non-parametric	Wilcoxon signed-rank test	r	✓
`ggwithinstats` (> 2 groups)	Non-parametric	Friedman rank sum test	$W_{Kendall}$	✓
`ggwithinstats` (2 groups)	Robust	Yuen's test on trimmed means for dependent samples	ξ	✓
`ggwithinstats` (> 2 groups)	Robust	Heteroscedastic one-way repeated measures ANOVA for trimmed means	×	×

Functions	Type	Test	Effect size	95% CI available?
`ggpiestats` and `ggbarstats` (unpaired)	Parametric	Pearson's χ^2 test	Cramér's V	✓
`ggpiestats` and `ggbarstats` (paired)	Parametric	McNemar's test	Cohen's g	✓
`ggpiestats`	Parametric	One-sample proportion test	Cramér's V	✓
`ggscatterstats` and `ggcorrmat`	Parametric	Pearson's r	r	✓
`ggscatterstats` and `ggcorrmat`	Non-parametric	Spearman's ρ	ρ	✓
`ggscatterstatsand` `ggcorrmat`	Robust	Percentage bend correlation	r	✓
`gghistostats` and `ggdotplotstats`	Parametric	One-sample t-test	Cohen's d, Hedge's g	✓
`gghistostats`	Non-parametric	One-sample Wilcoxon signed rank test	r	✓
`gghistostats` and `ggdotplotstats`	Robust	One-sample percentile bootstrap	robust estimator	✓
`ggcoefstats`	Parametric	Regression models	β	✓

4.4 Dealing with null results:

All functions therefore by default return Bayes Factor in favor of the null hypothesis by default. If the null hypothesis can't be rejected with the null hypothesis significance testing (NHST) approach, the Bayesian approach can help index evidence in favor of the null hypothesis (i.e., BF_01). By default, natural logarithms are shown because Bayes Factor values can sometimes be pretty large. Having values on logarithmic scale also makes it easy to compare evidence in favor alternative (BF_10) versus null (BF_01) hypotheses (since $log\_e(BF\_01) = -log\_e(BF\_01)$).

4.5 Avoiding the "p-value error":

The p-value indexes the probability that the researchers have falsely rejected a true null hypothesis (Type I error, i.e.) and can rarely be *exactly* 0. And yet over 97,000 manuscripts on Google Scholar report the p-value to be `p =0.000`, putatively due to relying on default computer software outputs (Lilienfeld et al., 2015). All p-values displayed in `ggstatsplot` plots avoid this mistake. Anything less than `p < 0.001` is displayed as such (e.g, Figure 1). The package deems it unimportant how infinitesimally small the p-values are and, instead, puts emphasis on the effect size magnitudes and their 95% CIs.

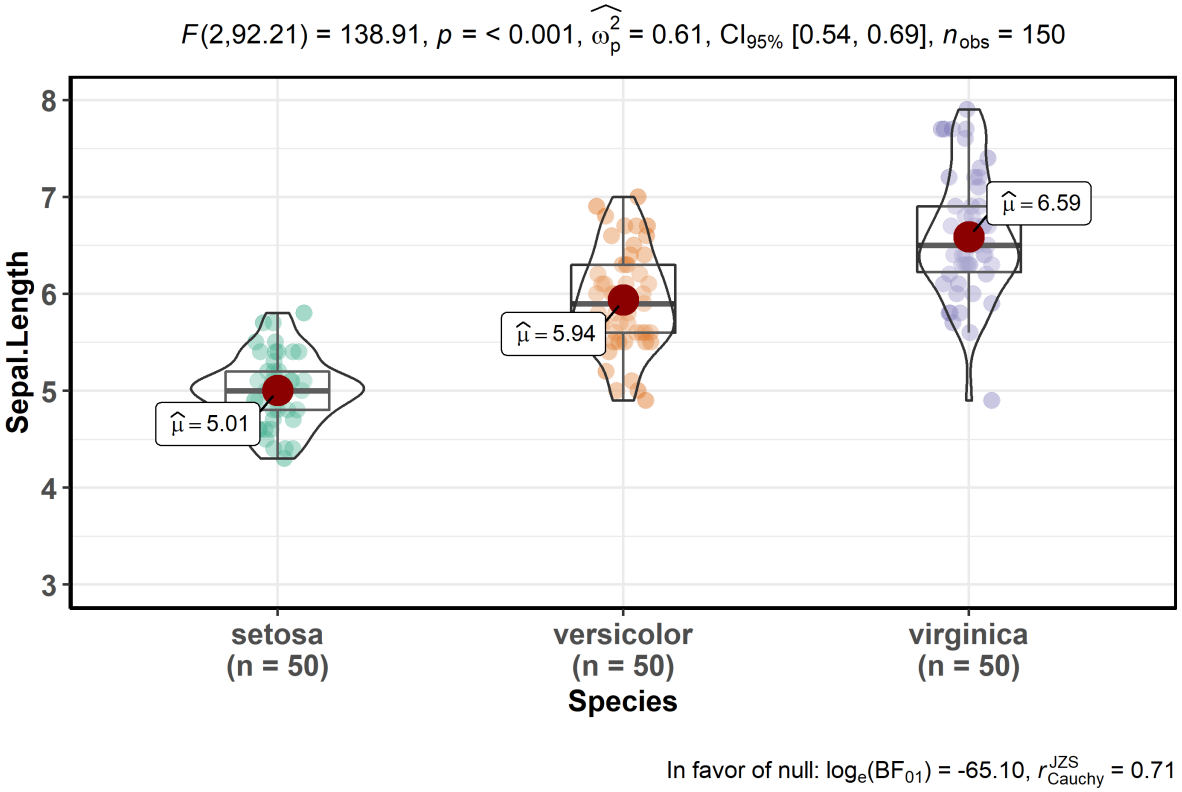
5 Short tutorial on primary functions

Here are examples of the main functions currently supported in `ggstatsplot`.

5.1 `ggbetweenstats`

This function creates either a violin plot, a box plot, or a mix of two for **between**-group or **between**-condition comparisons with results from statistical tests in the subtitle. The simplest function call looks like this-

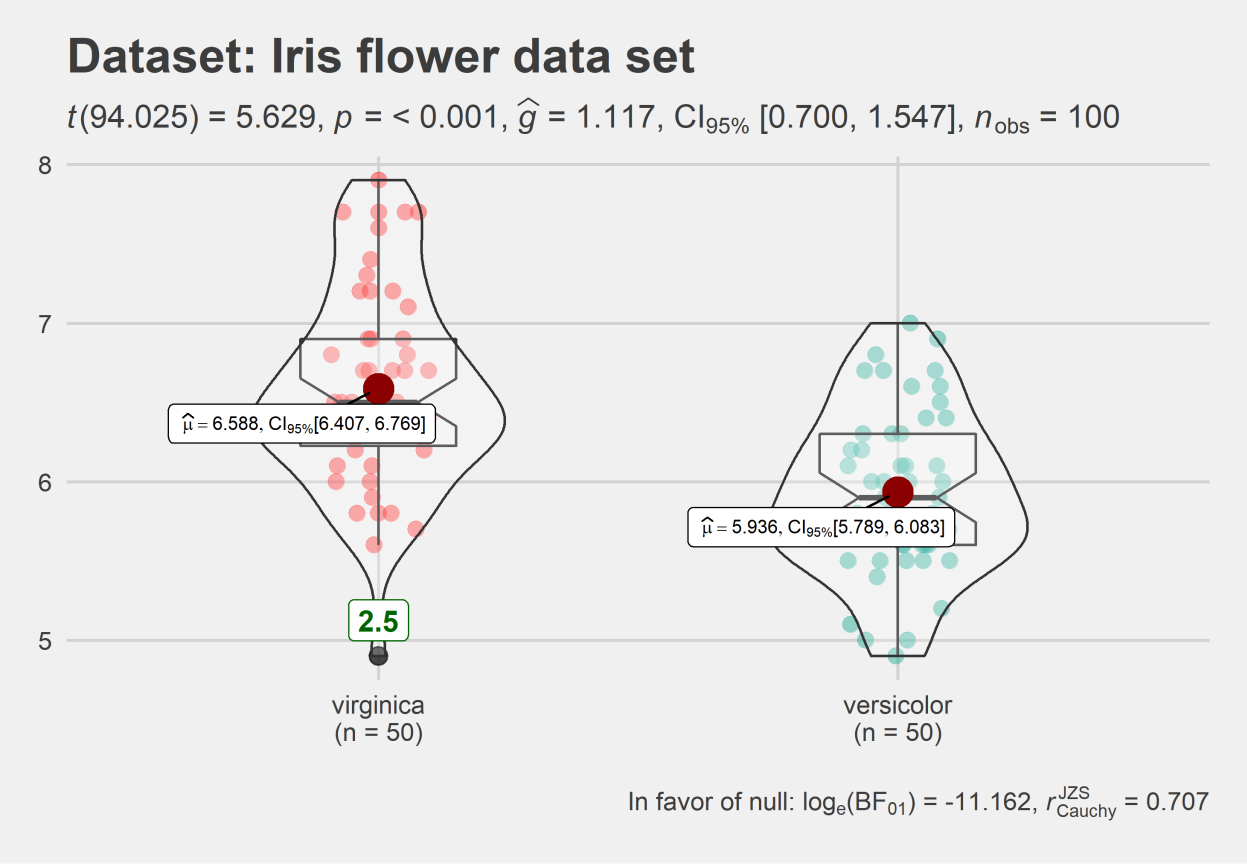
```
# loading needed libraries
library(ggstatsplot)

# for reproducibility
set.seed(123)

# plot
ggstatsplot::ggbetweenstats(
  data = iris,
  x = Species,
  y = Sepal.Length,
  messages = FALSE
) + # further modification outside of ggstatsplot
  ggplot2::coord_cartesian(ylim = c(3, 8)) +
  ggplot2::scale_y_continuous(breaks = seq(3, 8, by = 1))
```



Note that this function returns object of class `ggplot` and thus can be further modified using `ggplot2` functions.

A number of other arguments can be specified to make this plot even more informative or change some of the default options. Additionally, this time we will use a grouping variable that has only two levels. The function will automatically switch from carrying out an ANOVA analysis to a *t*-test.

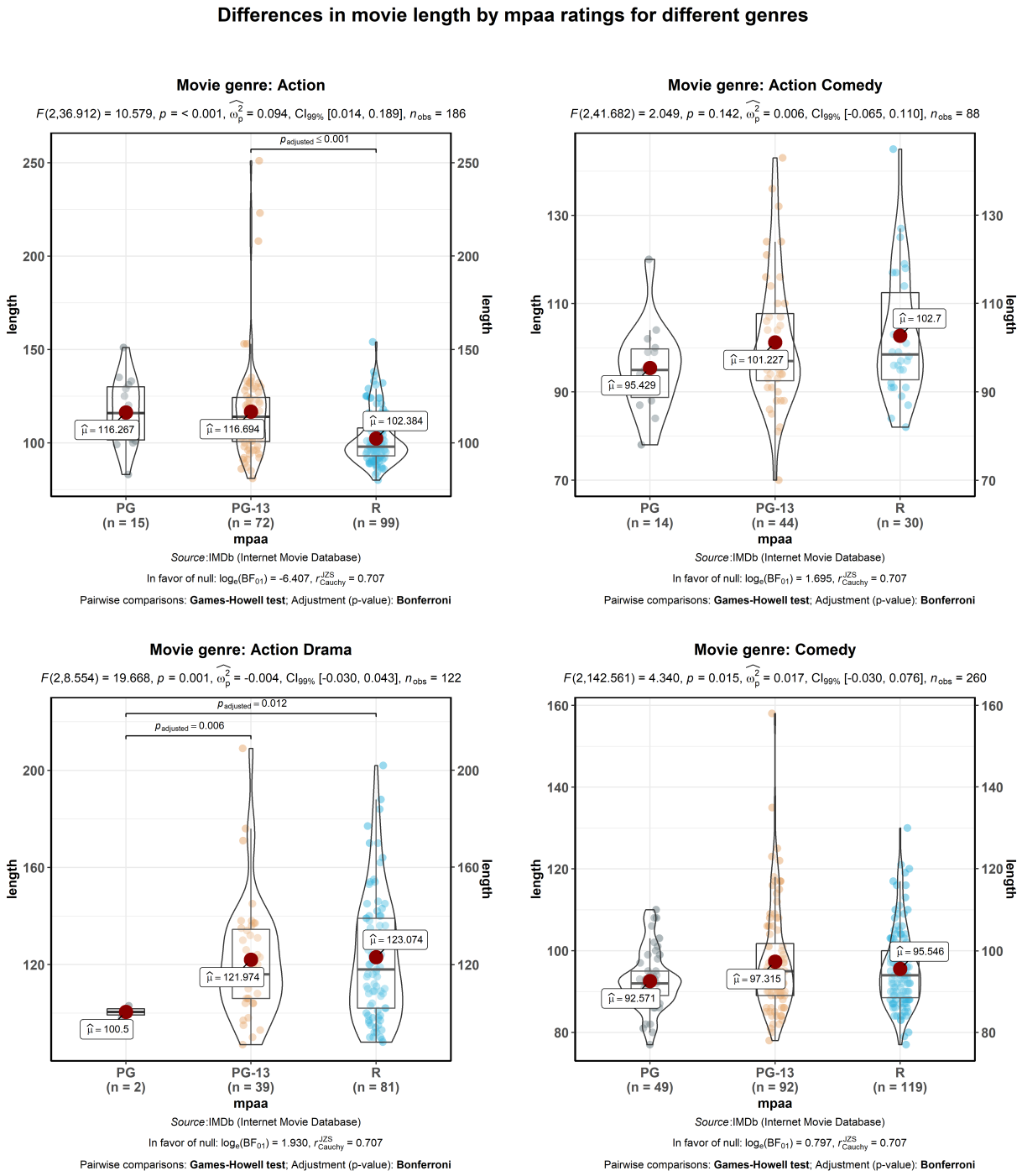
```
# for reproducibility
set.seed(123)
library(ggplot2)

# let's leave out one of the factor levels and see if instead of anova, a t-test will be run
iris2 <- dplyr::filter(.data = iris, Species != "setosa")

# let's change the levels of our factors, a common routine in data analysis
# pipeline, to see if this function respects the new factor levels
iris2$Species <- factor(x = iris2$Species, levels = c("virginica", "versicolor"))

# plot
ggstatsplot::ggbetweenstats(
  data = iris2,
  x = Species,
  y = Sepal.Length,
  notch = TRUE, # show notched box plot
  mean.plotting = TRUE, # whether mean for each group is to be displayed
  mean.ci = TRUE, # whether to display confidence interval for means
  mean.label.size = 2.5, # size of the label for mean
  type = "parametric", # which type of test is to be run
  k = 3, # number of decimal places for statistical results
  outlier.tagging = TRUE, # whether outliers need to be tagged
  outlier.label = Sepal.Width, # variable to be used for the outlier tag
  outlier.label.color = "darkgreen", # changing the color for the text label
  xlab = "Type of Species", # label for the x-axis variable
  ylab = "Attribute: Sepal Length", # label for the y-axis variable
  title = "Dataset: Iris flower data set", # title text for the plot
  ggtheme = ggthemes::theme_fivethirtyeight(), # choosing a different theme
  ggstatsplot.layer = FALSE, # turn off ggstatsplot theme layer
  package = "wesanderson", # package from which color palette is to be taken
  palette = "Darjeeling1", # choosing a different color palette
  messages = FALSE
)
```



Additionally, there is also a `grouped_` variant of this function that makes it easy to repeat the same operation across a **single** grouping variable:

```
# for reproducibility
set.seed(123)

# plot
ggstatsplot::grouped_ggbetweenstats(
  data = dplyr::filter(
    .data = ggstatsplot::movies_long,
    genre %in% c("Action", "Action Comedy", "Action Drama", "Comedy")
  ),
  x = mpaa,
  y = length,
  grouping.var = genre, # grouping variable
  pairwise.comparisons = TRUE, # display significant pairwise comparisons
  pairwise.annotation = "p.value", # how do you want to annotate the pairwise comparisons
  p.adjust.method = "bonferroni", # method for adjusting p-values for multiple comparisons
  conf.level = 0.99, # changing confidence level to 99%
  ggplot.component = list( # adding new components to `ggstatsplot` default
    ggplot2::scale_y_continuous(sec.axis = ggplot2::dup_axis())
  ),
  k = 3,
  title.prefix = "Movie genre",
  caption = substitute(paste(italic("Source"), ":IMDb (Internet Movie Database)")),
  palette = "default_jama",
```

```
  package = "ggsci",
  messages = FALSE,
  nrow = 2,
  title.text = "Differences in movie length by mpaa ratings for different genres"
)
```



5.1.1 Summary of tests

Following (between-subjects) tests are carried out for each type of analyses-

Type	No. of groups	Test
Parametric	> 2	Fisher's or Welch's one-way ANOVA
Non-parametric	> 2	Kruskal–Wallis one-way ANOVA
Robust	> 2	Heteroscedastic one-way ANOVA for trimmed means
Bayes Factor	> 2	Fisher's ANOVA
Parametric	2	Student's or Welch's t-test
Non-parametric	2	Mann–Whitney U test
Robust	2	Yuen's test for trimmed means
Bayes Factor	2	Student's t-test

The omnibus effect in one-way ANOVA design can also be followed up with more focal pairwise comparison tests. Here is a summary of *multiple pairwise comparison* tests supported in *ggbetweenstats*-

Type	Equal variance?	Test	p-value adjustment?
Parametric	No	Games-Howell test	Yes
Parametric	Yes	Student's t-test	Yes
Non-parametric	No	Dwass-Steel-Crichtlow-Fligner test	Yes
Robust	No	Yuen's trimmed means test	Yes
Bayes Factor	No	No	No
Bayes Factor	Yes	No	No

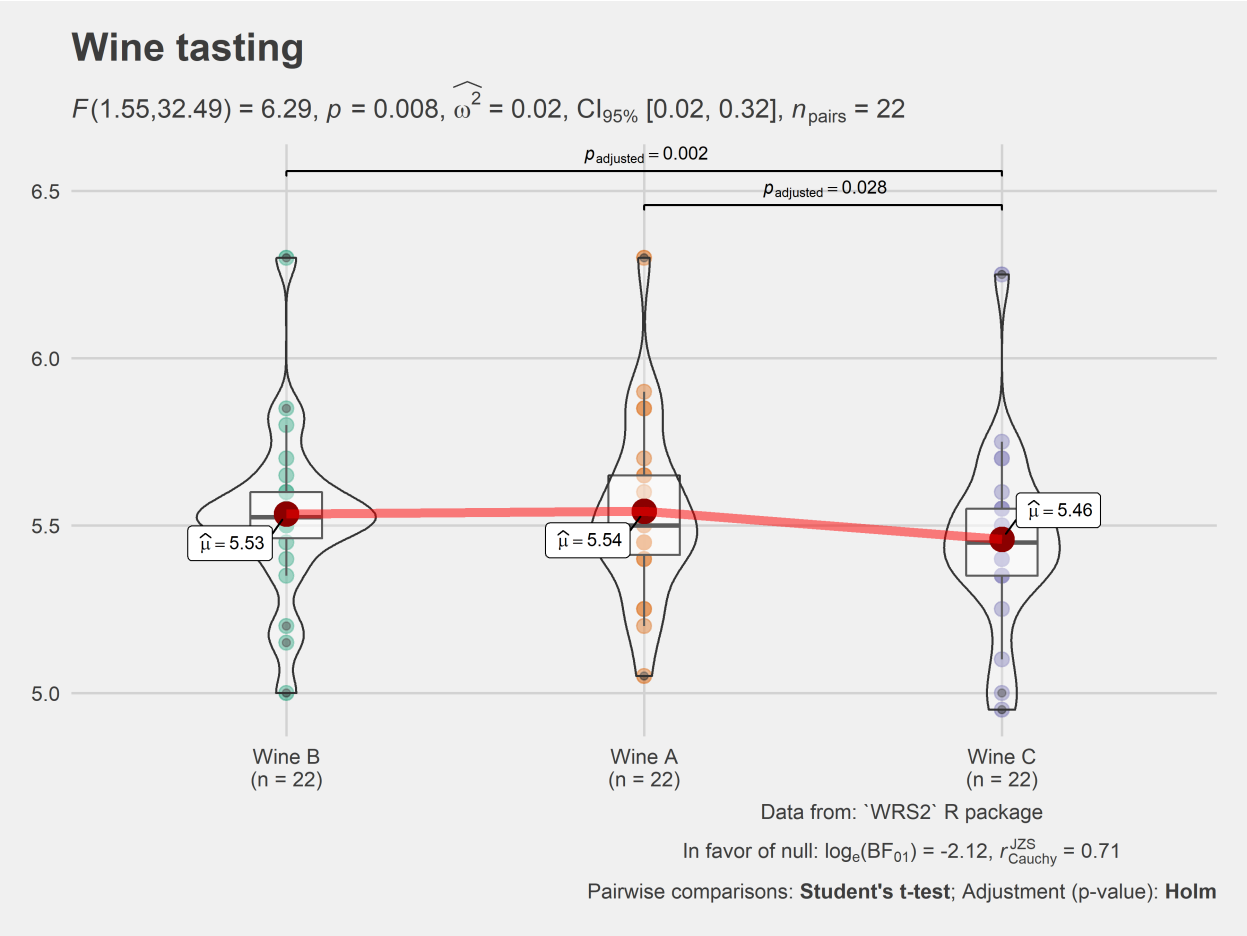
For more, see the `ggbetweenstats` vignette: https://indrajeetpatil.github.io/ggstatsplot/articles/web_only/ggbetweenstats.html

5.2 `ggwithinstats`

`ggbetweenstats` function has an identical twin function `ggwithinstats` for repeated measures designs that behaves in the same fashion with a few minor tweaks introduced to properly visualize the repeated measures design. As can be seen from an example below, the only difference between the plot structure is that now the group means are connected by paths to highlight the fact that these data are paired with each other.

```
# for reproducibility and data
set.seed(123)
library(WRS2)

# plot
ggstatsplot::ggwithinstats(
  data = WRS2::WineTasting,
  x = Wine,
  y = Taste,
  sort = "descending", # ordering groups along the x-axis based on
  sort.fun = median, # values of `y` variable
  pairwise.comparisons = TRUE,
  pairwise.display = "s",
  pairwise.annotation = "p",
  title = "Wine tasting",
```

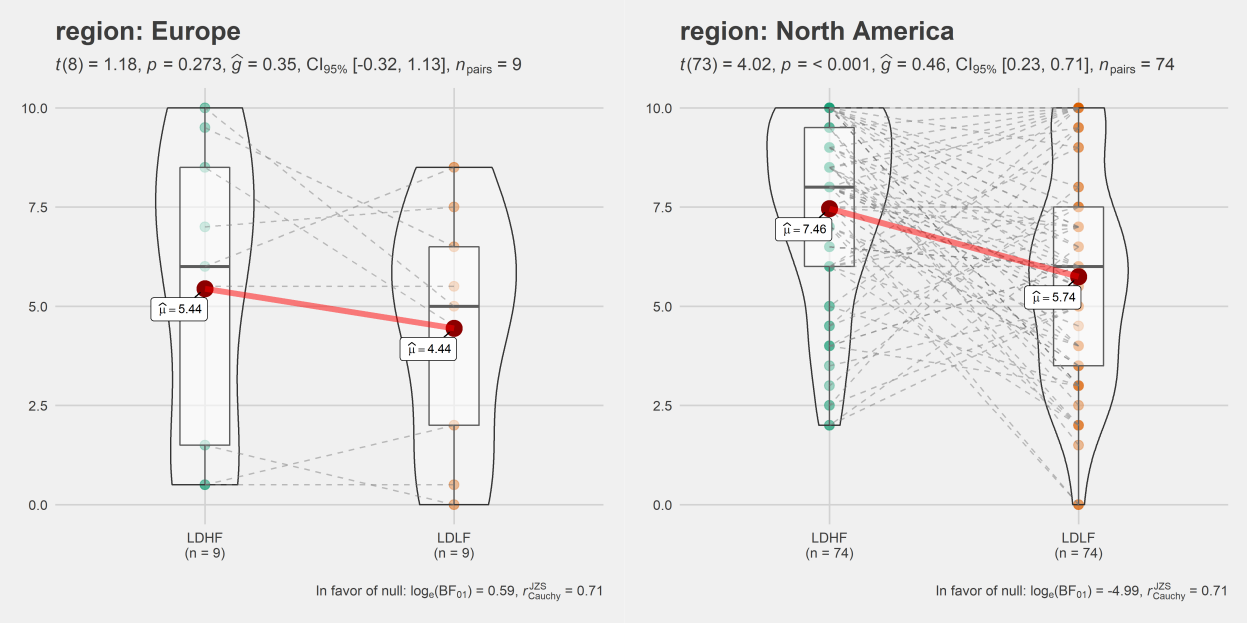
```
  caption = "Data from: `WRS2` R package",
  ggtheme = ggthemes::theme_fivethirtyeight(),
  ggstatsplot.layer = FALSE,
  messages = FALSE
)
```



As with the `ggbetweenstats`, this function also has a `grouped_` variant that makes repeating the same analysis across a single grouping variable quicker.

```
# common setup
set.seed(123)

# getting data in tidy format
data_bugs <- ggstatsplot::bugs_long %>%
  dplyr::filter(.data = ., region %in% c("Europe", "North America"))

# plot
ggstatsplot::grouped_ggwithinstats(
  data = dplyr::filter(data_bugs, condition %in% c("LDLF", "LDHF")),
  x = condition,
  y = desire,
  xlab = "Condition",
  ylab = "Desire to kill an artrhopod",
  grouping.var = region,
```

```
  outlier.tagging = TRUE,
  outlier.label = education,
  ggtheme = ggthemes::theme_fivethirtyeight(),
  ggstatsplot.layer = FALSE,
  messages = FALSE
)
```



5.2.1 Summary of tests

Following (within-subjects) tests are carried out for each type of analyses-

Type	No. of groups	Test
Parametric	> 2	One-way repeated measures ANOVA
Non-parametric	> 2	Friedman test
Robust	> 2	Heteroscedastic one-way repeated measures ANOVA for trimmed means
Bayes Factor	> 2	One-way repeated measures ANOVA
Parametric	2	Student's *t*-test
Non-parametric	2	Wilcoxon signed-rank test
Robust	2	Yuen's test on trimmed means for dependent samples
Bayes Factor	2	Student's *t*-test

The omnibus effect in one-way ANOVA design can also be followed up with more focal pairwise comparison tests. Here is a summary of *multiple pairwise comparison* tests supported in *ggwithinstats*-

Type	Test	*p*-value adjustment?
Parametric	Student's *t*-test	Yes
Non-parametric	Durbin-Conover test	Yes
Robust	Yuen's trimmed means test	Yes

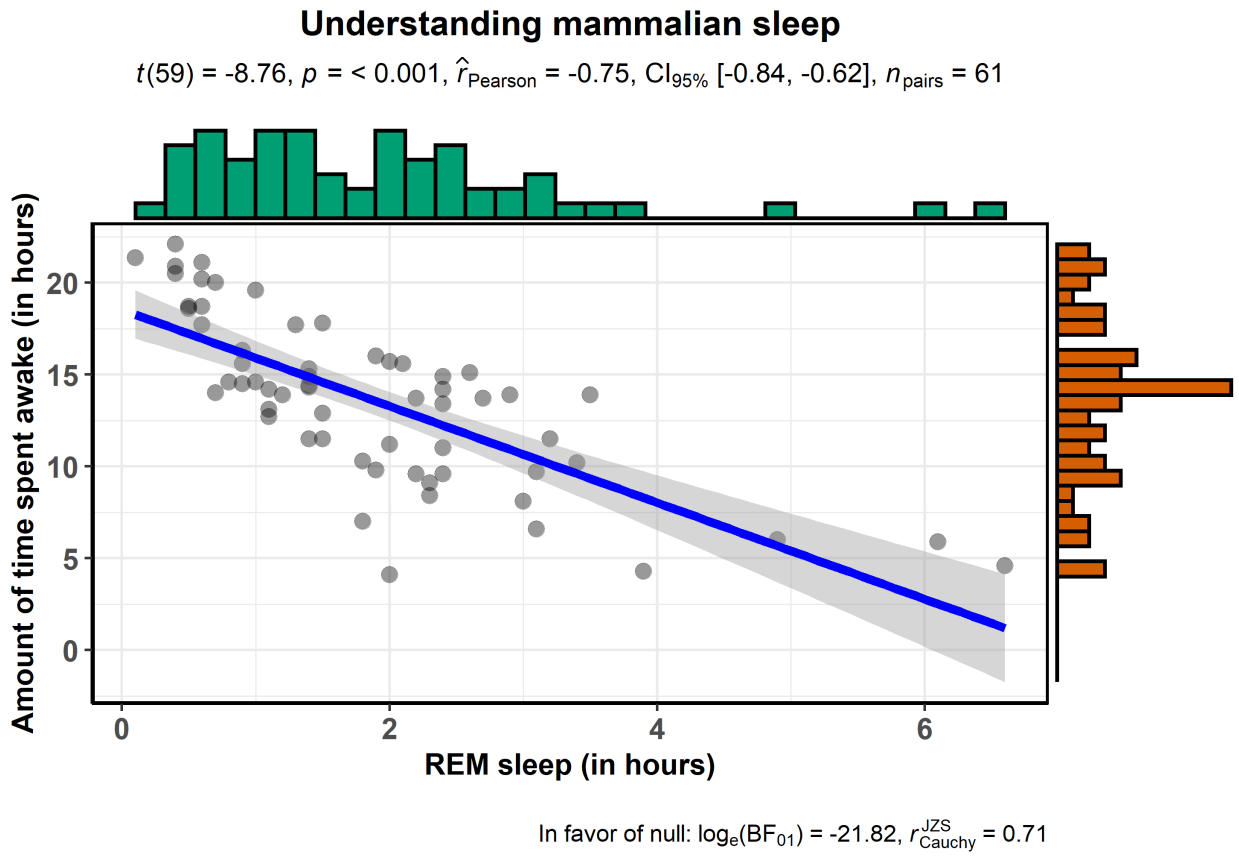
Type	Test	p-value adjustment?
Bayes Factor	No	No

For more, see the `ggwithinstats` vignette: https://indrajeetpatil.github.io/ggstatsplot/articles/web_only/ggwithinstats.html

5.3 `ggscatterstats`

This function creates a scatterplot with marginal distributions overlaid on the axes (from `ggExtra::ggMarginal`) and results from statistical tests in the subtitle:

```
ggstatsplot::ggscatterstats(
  data = ggplot2::msleep,
  x = sleep_rem,
  y = awake,
  xlab = "REM sleep (in hours)",
  ylab = "Amount of time spent awake (in hours)",
  title = "Understanding mammalian sleep",
  messages = FALSE
)
```



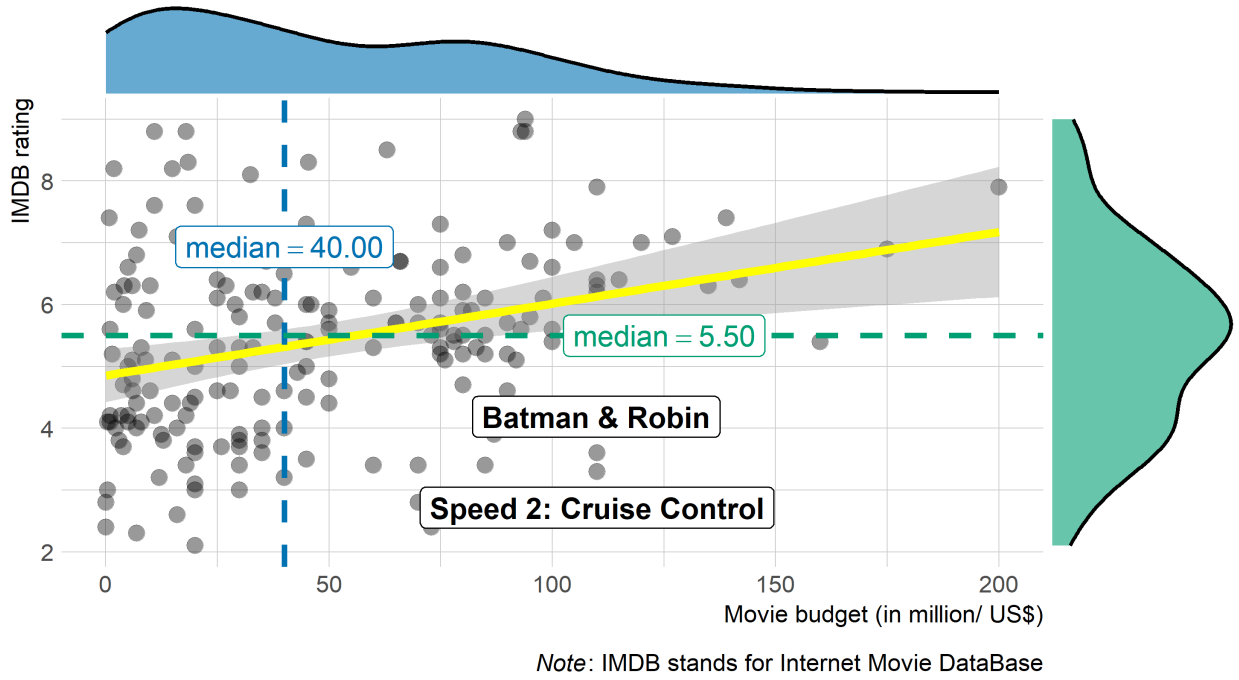
The available marginal distributions are-

- histograms

- boxplots
- density
- violin
- densigram (density + histogram)

Number of other arguments can be specified to modify this basic plot-

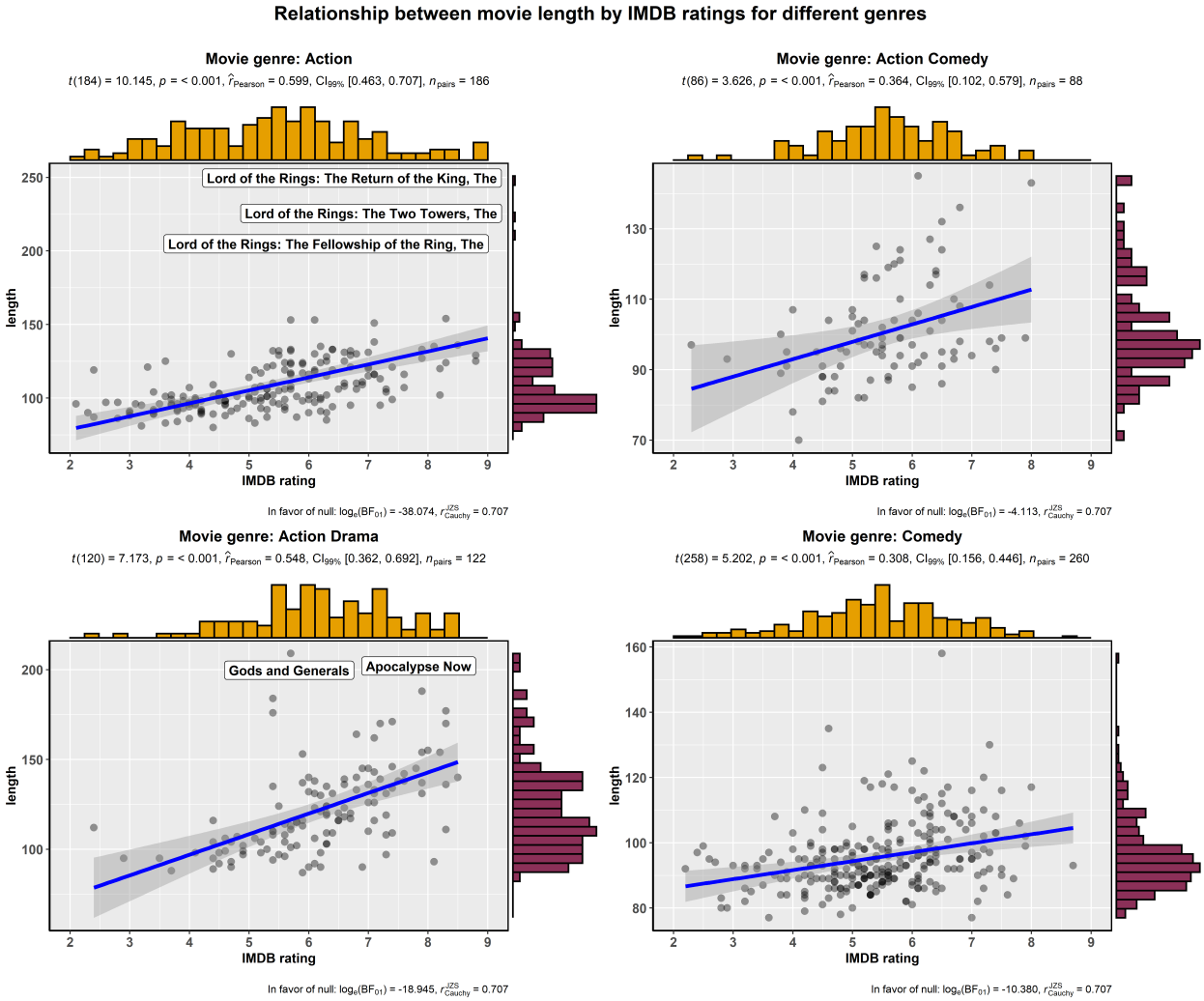
```
# for reproducibility
set.seed(123)

# plot
ggstatsplot::ggscatterstats(
  data = dplyr::filter(.data = ggstatsplot::movies_long, genre == "Action"),
  x = budget,
  y = rating,
  type = "robust", # type of test that needs to be run
  conf.level = 0.99, # confidence level
  xlab = "Movie budget (in million/ US$)", # label for x axis
  ylab = "IMDB rating", # label for y axis
  label.var = "title", # variable for labeling data points
  label.expression = "rating < 5 & budget > 100", # expression that decides which points to label
  line.color = "yellow", # changing regression line color line
  title = "Movie budget and IMDB rating (action)", # title text for the plot
  caption = expression( # caption text for the plot
    paste(italic("Note"), ": IMDB stands for Internet Movie DataBase")
  ),
  ggtheme = hrbrthemes::theme_ipsum_ps(), # choosing a different theme
  ggstatsplot.layer = FALSE, # turn off ggstatsplot theme layer
  marginal.type = "density", # type of marginal distribution to be displayed
  xfill = "#0072B2", # color fill for x-axis marginal distribution
  yfill = "#009E73", # color fill for y-axis marginal distribution
  xalpha = 0.6, # transparency for x-axis marginal distribution
  yalpha = 0.6, # transparency for y-axis marginal distribution
  centrality.para = "median", # central tendency lines to be displayed
  messages = FALSE # turn off messages and notes
)
```



Additionally, there is also a `grouped_` variant of this function that makes it easy to repeat the same operation across a **single** grouping variable. Also, note that, as opposed to the other functions, this function does not return a `ggplot` object and any modification you want to make can be made in advance using `ggplot.component` argument (available for all functions, but especially useful for this particular function):

```
# for reproducibility
set.seed(123)

# plot
ggstatsplot::grouped_ggscatterstats(
  data = dplyr::filter(
    .data = ggstatsplot::movies_long,
    genre %in% c("Action", "Action Comedy", "Action Drama", "Comedy")
  ),
  x = rating,
  y = length,
  label.var = title,
  label.expression = length > 200,
  conf.level = 0.99,
  k = 3, # no. of decimal places in the results
  xfill = "#E69F00",
  yfill = "#8b3058",
  xlab = "IMDB rating",
  grouping.var = genre, # grouping variable
  title.prefix = "Movie genre",
  ggtheme = ggplot2::theme_grey(),
```

```
  ggplot.component = list(
    ggplot2::scale_x_continuous(breaks = seq(2, 9, 1), limits = (c(2, 9)))
  ),
  messages = FALSE,
  nrow = 2,
  title.text = "Relationship between movie length by IMDB ratings for different genres"
)
```



5.3.1 Summary of tests

Following tests are carried out for each type of analyses. Additionally, the correlation coefficients (and their confidence intervals) are used as effect sizes-

Type	Test	CI?
Parametric	Pearson's correlation coefficient	Yes
Non-parametric	Spearman's rank correlation coefficient	Yes
Robust	Percentage bend correlation coefficient	Yes
Bayes Factor	Pearson's correlation coefficient	No

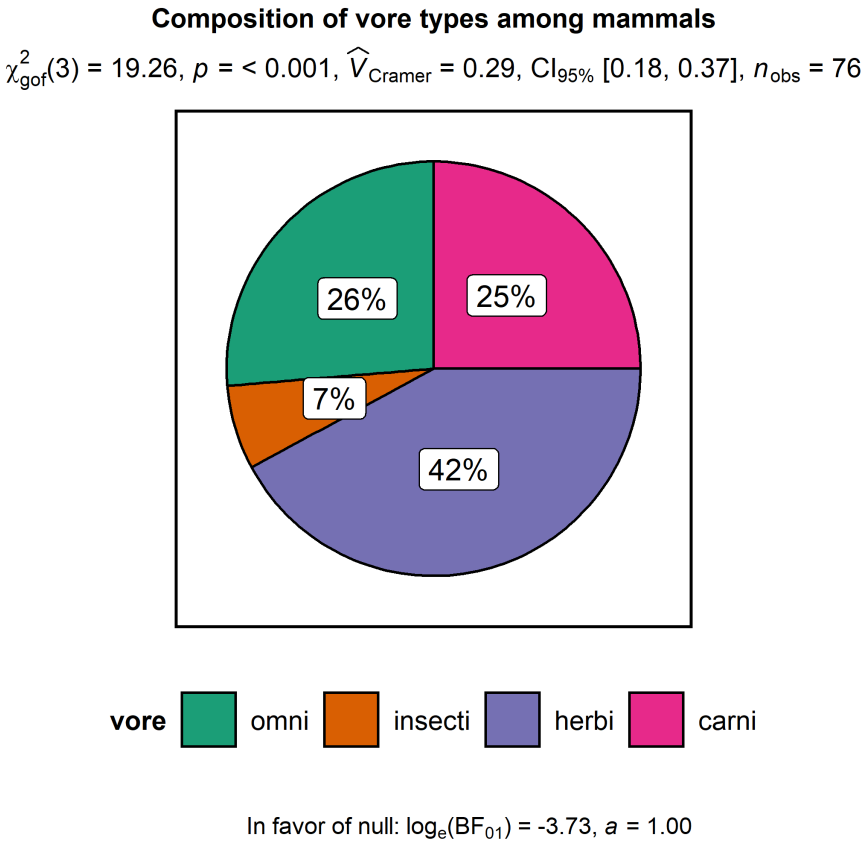
For more, see the `ggscatterstats` vignette: https://indrajeetpatil.github.io/ggstatsplot/articles/web_only/ggscatterstats.html

5.4 ggpiestats

This function creates a pie chart for categorical or nominal variables with results from contingency table analysis (Pearson's χ^2 test for between-subjects design and McNemar's χ^2 test for within-subjects design) included in the subtitle of the plot. If only one categorical variable is entered, results from one-sample proportion test (i.e., a χ^2 goodness of fit/gof test) will be displayed as a subtitle.

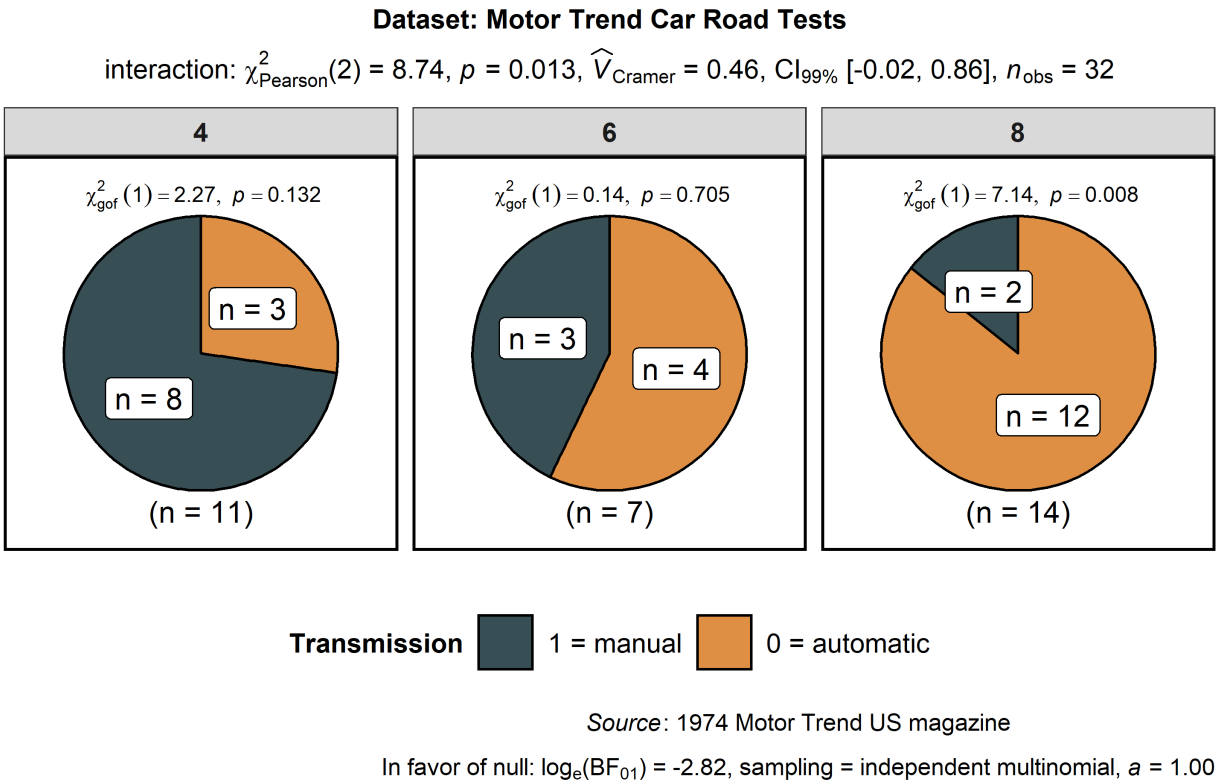
Here is an example of a case where the theoretical question is about proportions for different levels of a single nominal variable:

```
# for reproducibility
set.seed(123)

# plot
ggstatsplot::ggpiestats(
  data = ggplot2::msleep,
  x = vore,
  title = "Composition of vore types among mammals",
  messages = FALSE
)
```



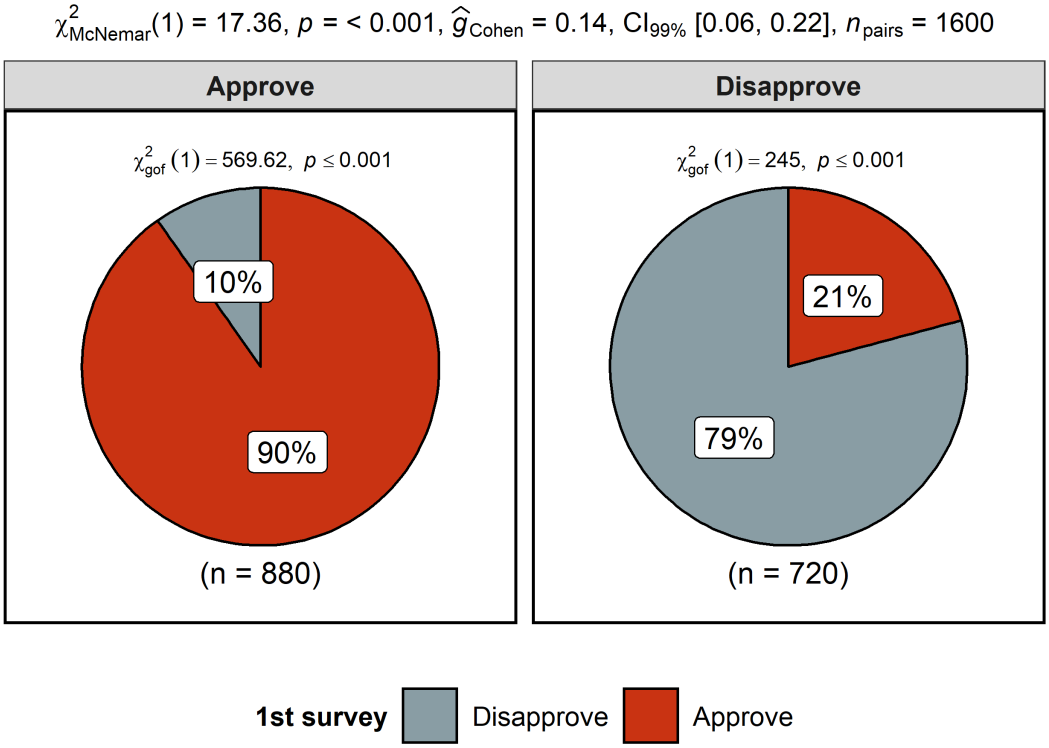
This function can also be used to study an interaction between two categorical variables:

```r
# for reproducibility
set.seed(123)

# plot
ggstatsplot::ggpiestats(
  data = mtcars,
  x = am,
  y = cyl,
  conf.level = 0.99, # confidence interval for effect size measure
  title = "Dataset: Motor Trend Car Road Tests", # title for the plot
  stat.title = "interaction: ", # title for the results
  legend.title = "Transmission", # title for the legend
  factor.levels = c("1 = manual", "0 = automatic"), # renaming the factor level names (`x`)
  facet.wrap.name = "No. of cylinders", # name for the facetting variable
  slice.label = "counts", # show counts data instead of percentages
  package = "ggsci", # package from which color palette is to be taken
  palette = "default_jama", # choosing a different color palette
  caption = substitute( # text for the caption
    paste(italic("Source"), ": 1974 Motor Trend US magazine")
  ),
  messages = FALSE # turn off messages and notes
)
```



In case of repeated measures designs, setting `paired = TRUE` will produce results from McNemar's χ^2 test-

```r
# for reproducibility
set.seed(123)

# data
survey.data <- data.frame(
  `1st survey` = c("Approve", "Approve", "Disapprove", "Disapprove"),
  `2nd survey` = c("Approve", "Disapprove", "Approve", "Disapprove"),
  `Counts` = c(794, 150, 86, 570),
  check.names = FALSE
)

# plot
ggstatsplot::ggpiestats(
  data = survey.data,
  x = `1st survey`,
  y = `2nd survey`,
  counts = Counts,
  paired = TRUE, # within-subjects design
  conf.level = 0.99, # confidence interval for effect size measure
  package = "wesanderson",
  palette = "Royal1"
)
#> Note: 99% CI for effect size estimate was computed with 100 bootstrap samples.
#> # A tibble: 2 x 11
#>   `2nd survey` counts  perc N         Approve Disapprove statistic   p.value
#>   <fct>         <int> <dbl> <chr>     <chr>   <chr>          <dbl>     <dbl>
#> 1 Disapprove      720   45  (n = 720) 20.83%  79.17%          245  3.20e- 55
#> 2 Approve         880   55. (n = 880) 90.23%  9.77%           570. 6.80e-126
#>   parameter method                                  significance
#>       <dbl> <chr>                                   <chr>
#> 1         1 Chi-squared test for given probabilities ***
#> 2         1 Chi-squared test for given probabilities ***
```



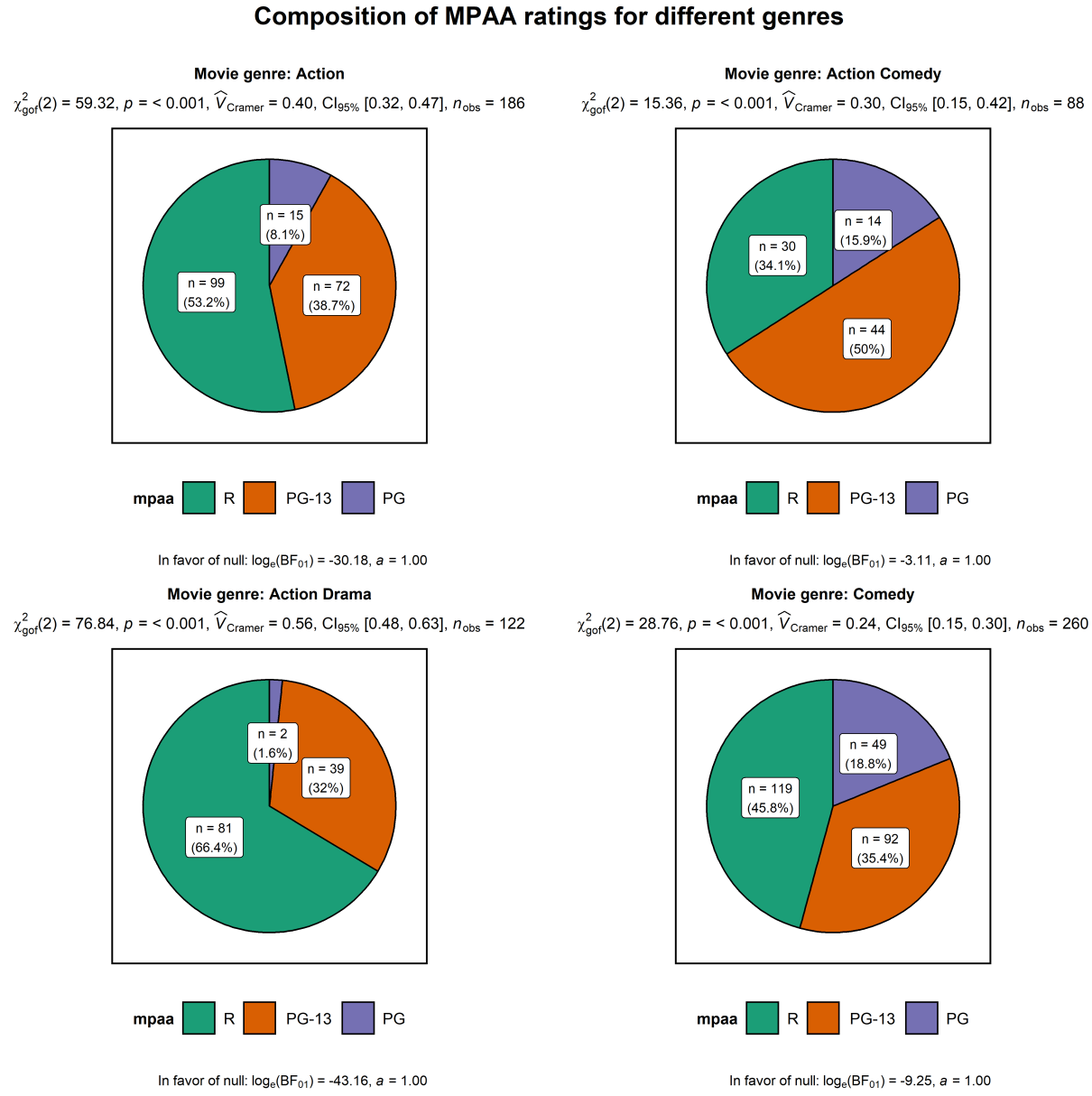
Note that when a two-way table is present (i.e., when both `x` and `y` arguments are specified), p-values for results from one-sample proportion tests are displayed in each facet in the form of asterisks with the following convention:

- $***$: $p < 0.001$
- $**$: $p < 0.01$
- $*$: $p < 0.05$
- ns: $p > 0.05$

Additionally, there is also a `grouped_` variant of this function that makes it easy to repeat the same operation across a **single** grouping variable:

```
# for reproducibility
set.seed(123)

# plot
ggstatsplot::grouped_ggpiestats(
  dplyr::filter(
    .data = ggstatsplot::movies_long,
    genre %in% c("Action", "Action Comedy", "Action Drama", "Comedy")
  ),
  x = mpaa,
  grouping.var = genre, # grouping variable
  title.prefix = "Movie genre", # prefix for the facetted title
  label.text.size = 3, # text size for slice labels
  slice.label = "both", # show both counts and percentage data
```

```
  perc.k = 1, # no. of decimal places for percentages
  messages = FALSE,
  nrow = 2,
  title.text = "Composition of MPAA ratings for different genres"
)
```



5.4.1 Summary of tests

Following tests are carried out for each type of analyses-

Type of data	Design	Test
Unpaired	$n \times p$ contingency table	Pearson's χ^2 test

Type of data	Design	Test
Paired	$n \times p$ contingency table	McNemar's χ^2 test
Frequency	$n \times 1$ contingency table	Goodness of fit (χ^2)

Following effect sizes (and confidence intervals/CI) are available for each type of test-

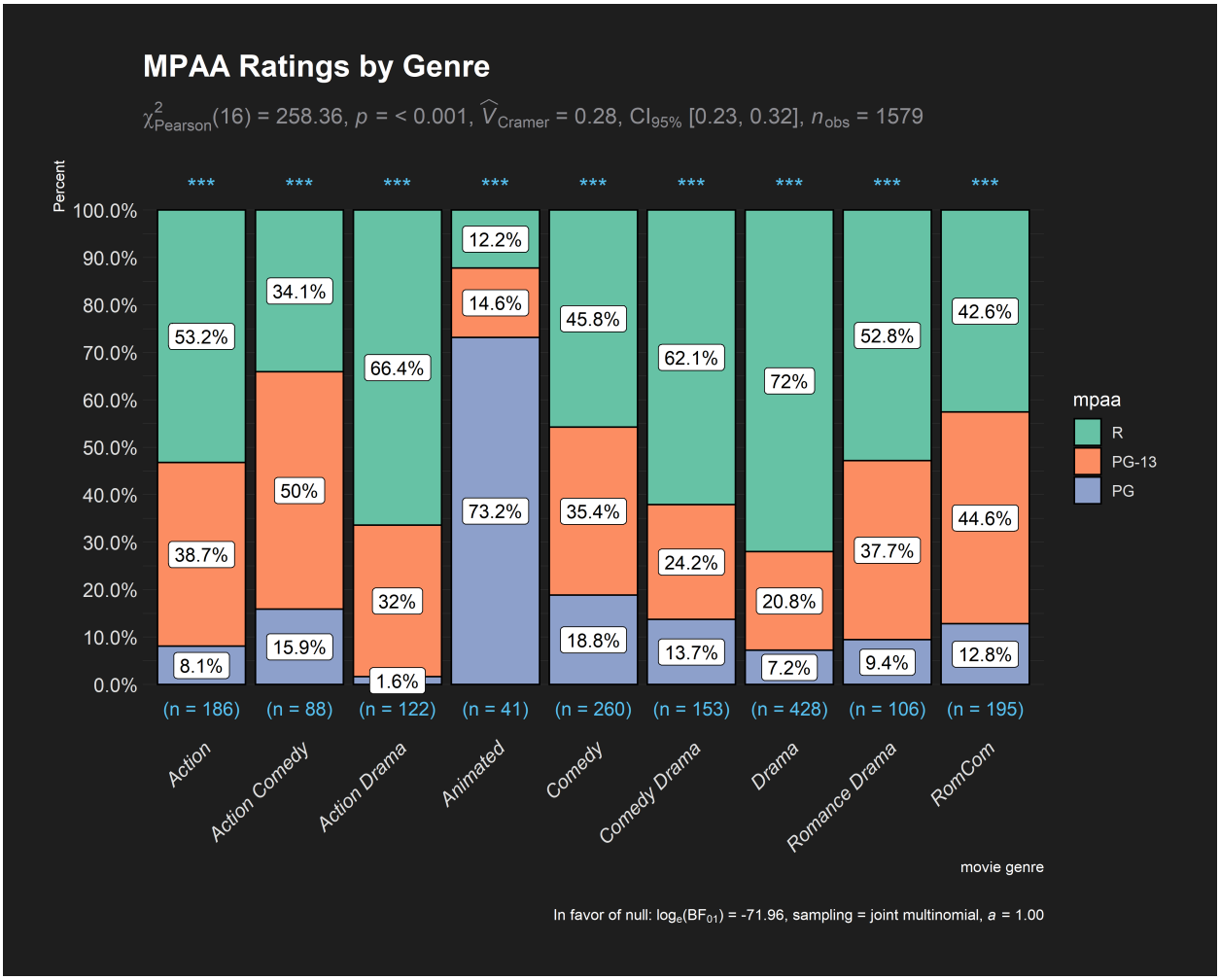
Type	Effect size	CI?
Pearson's chi-squared test	Cramér's V	Yes
McNemar's test	Cohen's g	Yes
Goodness of fit	Cramér's V	Yes

For more, see the `ggpiestats` vignette: https://indrajeetpatil.github.io/ggstatsplot/articles/web_only/ggpiestats.html

5.5 `ggbarstats`

In case you are not a fan of pie charts (for very good reasons), you can alternatively use `ggbarstats` function which has a similar syntax-

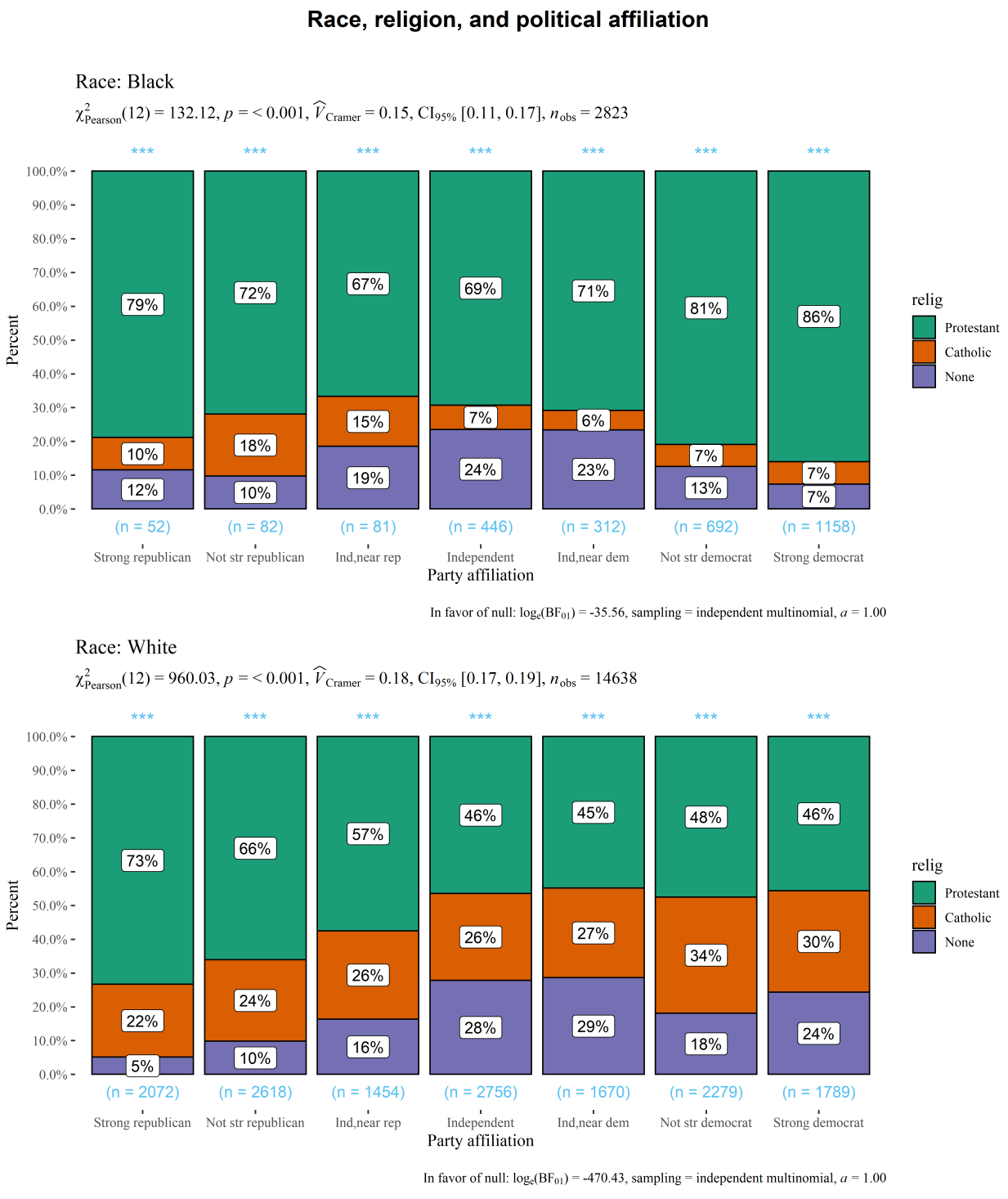
```
# for reproducibility
set.seed(123)

# plot
ggstatsplot::ggbarstats(
  data = ggstatsplot::movies_long,
  x = mpaa,
  y = genre,
  sampling.plan = "jointMulti",
  title = "MPAA Ratings by Genre",
  xlab = "movie genre",
  perc.k = 1,
  x.axis.orientation = "slant",
  ggtheme = hrbrthemes::theme_modern_rc(),
  ggstatsplot.layer = FALSE,
  ggplot.component = ggplot2::theme(axis.text.x = ggplot2::element_text(face = "italic")),
  palette = "Set2",
  messages = FALSE
)
```



And, needless to say, there is also a `grouped_` variant of this function-

```
# setup
set.seed(123)

# smaller dataset
df <- dplyr::filter(
  .data = forcats::gss_cat,
  race %in% c("Black", "White"),
  relig %in% c("Protestant", "Catholic", "None"),
  !partyid %in% c("No answer", "Don't know", "Other party")
)

# plot
ggstatsplot::grouped_ggbarstats(
  data = df,
  x = relig,
  y = partyid,
  grouping.var = race,
  title.prefix = "Race",
  xlab = "Party affiliation",
  ggtheme = ggthemes::theme_tufte(base_size = 12),
```

```
  ggstatsplot.layer = FALSE,
  messages = FALSE,
  title.text = "Race, religion, and political affiliation",
  nrow = 2
)
```



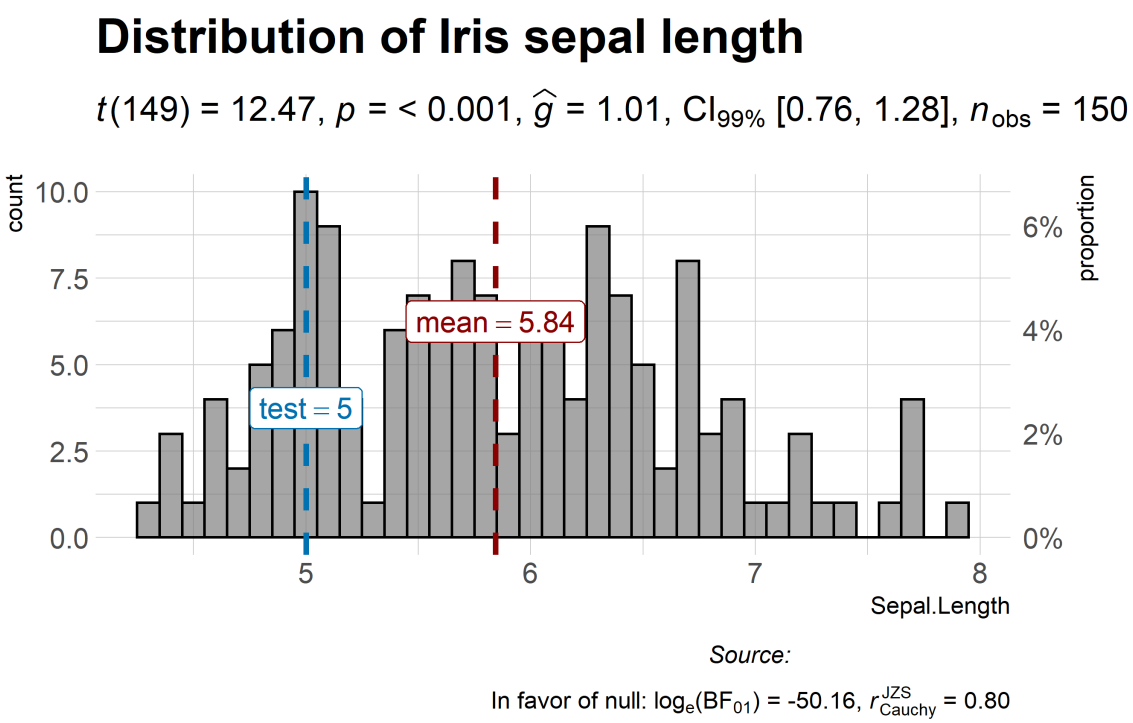
5.5.1 Summary of tests

This is identical to the `ggpiestats` function summary of tests.

5.6 gghistostats

In case you would like to see the distribution of a single variable and check if it is significantly different from a specified value with a one sample test, this function will let you do that.

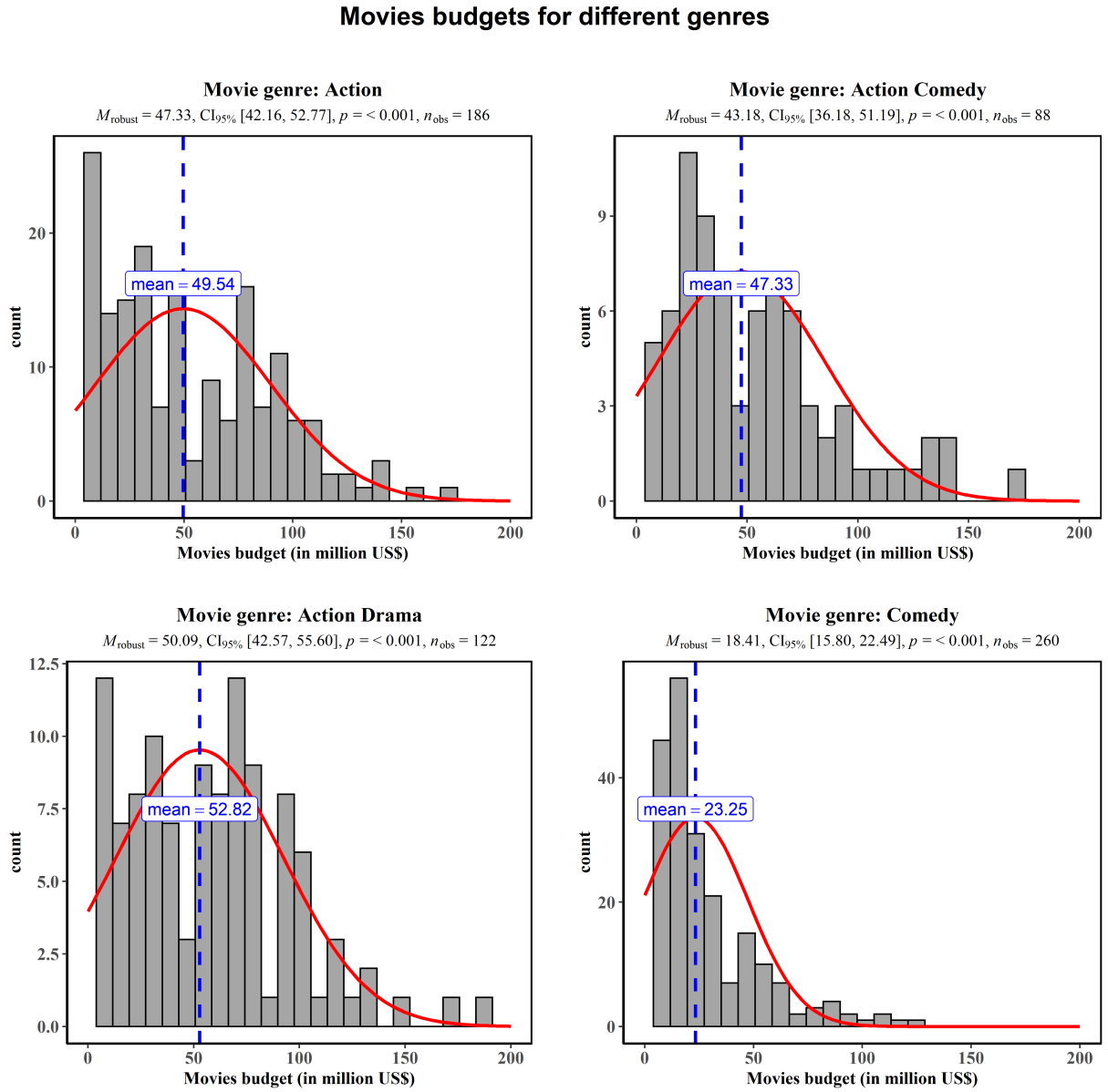
```
# for reproducibility
set.seed(123)

# plot
ggstatsplot::gghistostats(
  data = iris, # dataframe from which variable is to be taken
  x = Sepal.Length, # numeric variable whose distribution is of interest
  title = "Distribution of Iris sepal length", # title for the plot
  caption = substitute(paste(italic("Source:", "Ronald Fisher's Iris data set"))),
  type = "parametric", # one sample t-test
  conf.level = 0.99, # changing confidence level for effect size
  bar.measure = "mix", # what does the bar length denote
  test.value = 5, # default value is 0
  test.value.line = TRUE, # display a vertical line at test value
  test.value.color = "#0072B2", # color for the line for test value
  centrality.para = "mean", # which measure of central tendency is to be plotted
  centrality.color = "darkred", # decides color for central tendency line
  binwidth = 0.10, # binwidth value (experiment)
  bf.prior = 0.8, # prior width for computing bayes factor
  messages = FALSE, # turn off the messages
  ggtheme = hrbrthemes::theme_ipsum_tw(), # choosing a different theme
  ggstatsplot.layer = FALSE # turn off ggstatsplot theme layer
)
```



Additionally, there is also a `grouped_` variant of this function that makes it easy to repeat the same operation across a **single** grouping variable:

```
# for reproducibility
set.seed(123)

# plot
ggstatsplot::grouped_gghistostats(
  data = dplyr::filter(
    .data = ggstatsplot::movies_long,
    genre %in% c("Action", "Action Comedy", "Action Drama", "Comedy")
  ),
  x = budget,
  xlab = "Movies budget (in million US$)",
  type = "robust", # use robust location measure
  grouping.var = genre, # grouping variable
  normal.curve = TRUE, # superimpose a normal distribution curve
  normal.curve.color = "red",
  title.prefix = "Movie genre",
  ggtheme = ggthemes::theme_tufte(),
  ggplot.component = list( # modify the defaults from `ggstatsplot` for each plot
    ggplot2::scale_x_continuous(breaks = seq(0, 200, 50), limits = (c(0, 200)))
  ),
  messages = FALSE,
  nrow = 2,
  title.text = "Movies budgets for different genres"
```

```
)
```



5.6.1 Summary of tests

Following tests are carried out for each type of analyses-

Type	Test
Parametric	One-sample Student's *t*-test
Non-parametric	One-sample Wilcoxon test
Robust	One-sample percentile bootstrap
Bayes Factor	One-sample Student's *t*-test

Following effect sizes (and confidence intervals/CI) are available for each type of test-

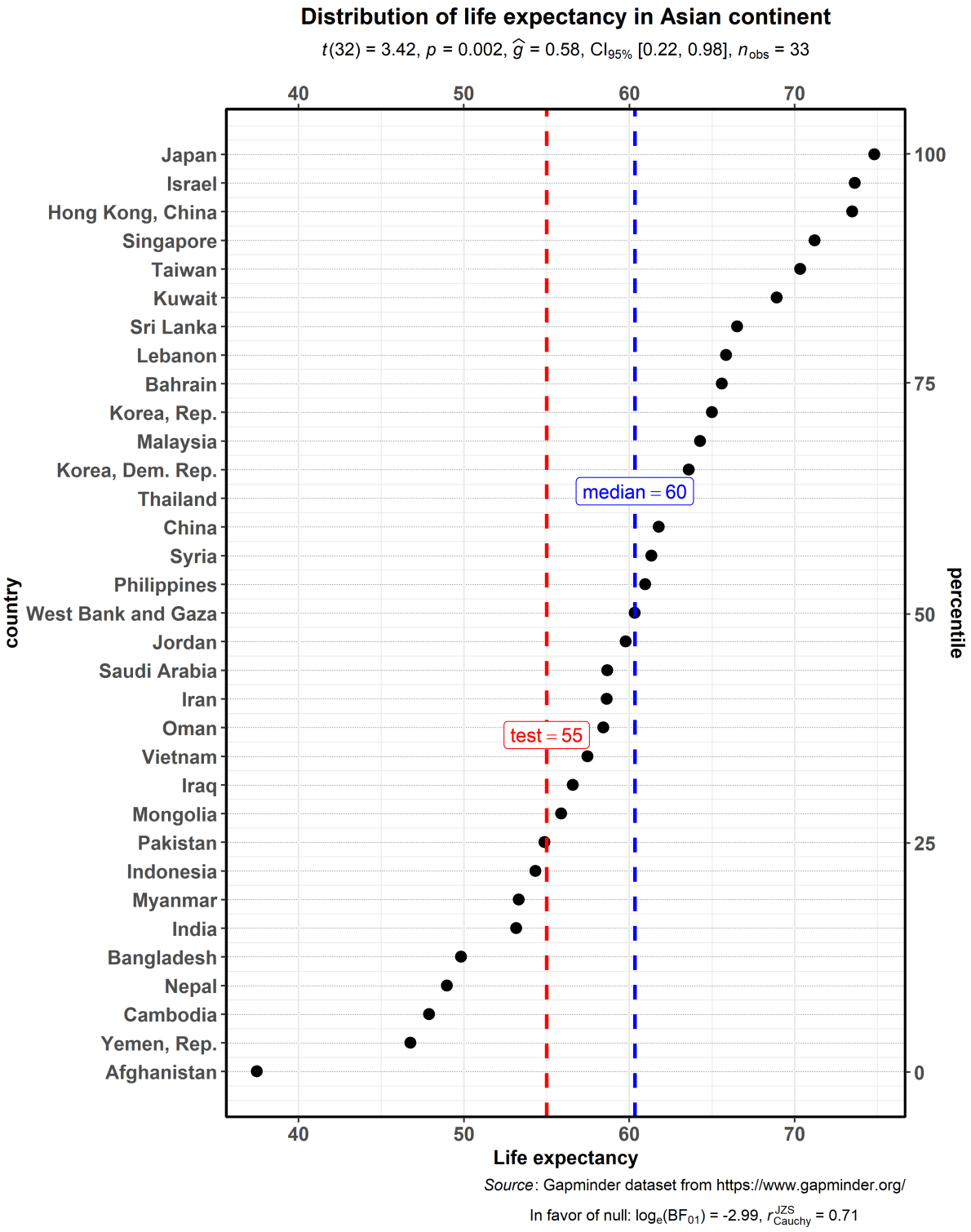
Type	Effect size	CI?
Parametric	Cohen's d, Hedge's g (central-and noncentral-t distribution based)	Yes
Non-parametric	r (computed as $Z/\sqrt{N}$)	Yes
Robust	M_{robust} (Robust location measure)	Yes
Bayes Factor	No	No

For more, including information about the variant of this function `grouped_gghistostats`, see the `gghistostats` vignette: https://indrajeetpatil.github.io/ggstatsplot/articles/web_only/gghistostats.html

5.7 `ggdotplotstats`

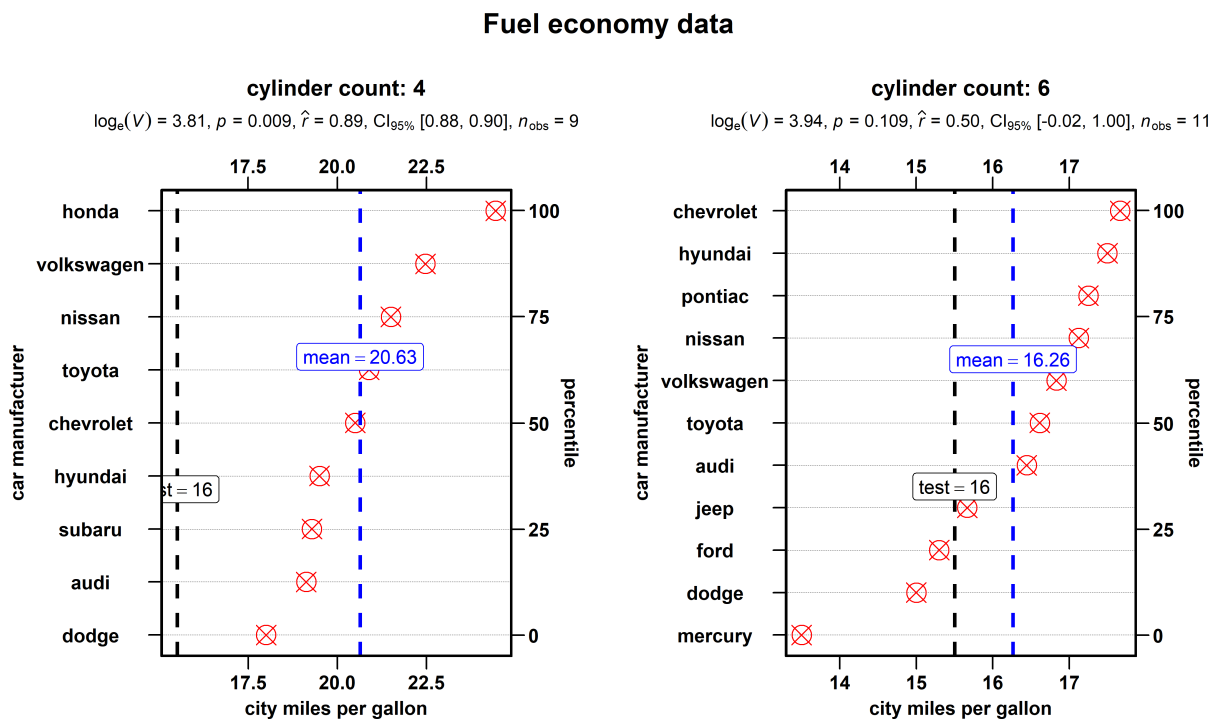
This function is similar to `gghistostats`, but is intended to be used when the numeric variable also has a label.

```
# for reproducibility
set.seed(123)

# plot
ggdotplotstats(
  data = dplyr::filter(.data = gapminder::gapminder, continent == "Asia"),
  y = country,
  x = lifeExp,
  test.value = 55,
  test.value.line = TRUE,
  test.line.labeller = TRUE,
  test.value.color = "red",
  centrality.para = "median",
  centrality.k = 0,
  title = "Distribution of life expectancy in Asian continent",
  xlab = "Life expectancy",
  messages = FALSE,
  caption = substitute(
    paste(
      italic("Source"),
      ": Gapminder dataset from https://www.gapminder.org/"
    )
  )
)
```



As with the rest of the functions in this package, there is also a `grouped_` variant of this function to facilitate looping the same operation for all levels of a single grouping variable.

```
# for reproducibility
set.seed(123)

# removing factor level with very few no. of observations
df <- dplyr::filter(.data = ggplot2::mpg, cyl %in% c("4", "6"))

# plot
ggstatsplot::grouped_ggdotplotstats(
  data = df,
  x = cty,
  y = manufacturer,
  xlab = "city miles per gallon",
  ylab = "car manufacturer",
  type = "nonparametric", # non-parametric test
  grouping.var = cyl, # grouping variable
  test.value = 15.5,
  title.prefix = "cylinder count",
  point.color = "red",
  point.size = 5,
  point.shape = 13,
  test.value.line = TRUE,
  ggtheme = ggthemes::theme_par(),
  messages = FALSE,
  title.text = "Fuel economy data"
)
```



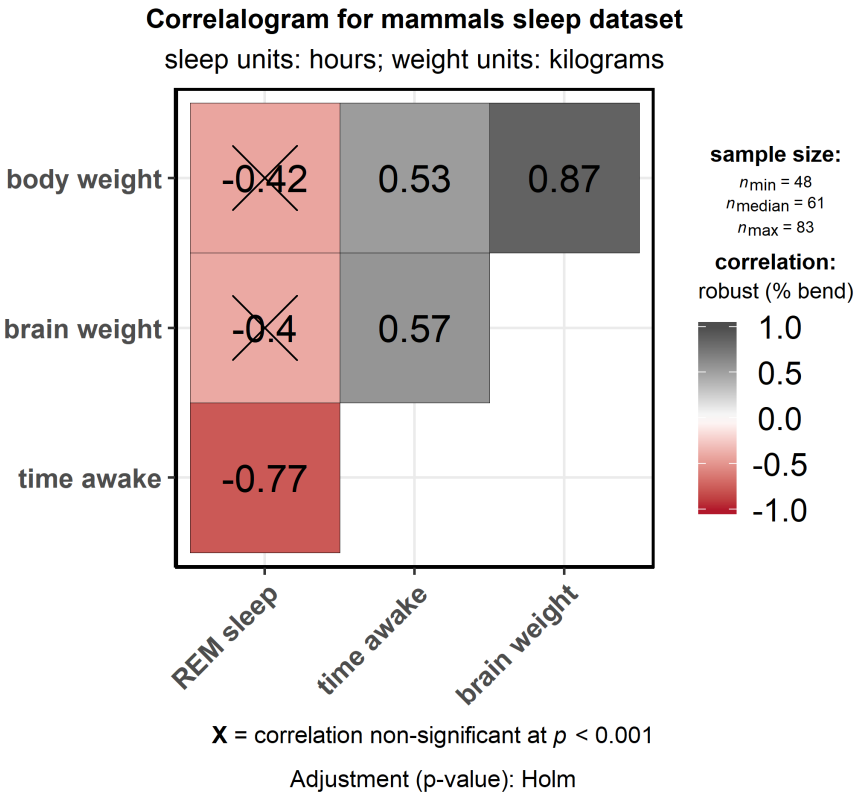
5.7.1 Summary of tests

This is identical to summary of tests for `gghistostats`.

5.8 ggcorrmat

`ggcorrmat` makes a correlalogram (a matrix of correlation coefficients) with minimal amount of code. Just sticking to the defaults itself produces publication-ready correlation matrices. But, for the sake of exploring the available options, let's change some of the defaults. For example, multiple aesthetics-related arguments can be modified to change the appearance of the correlation matrix.

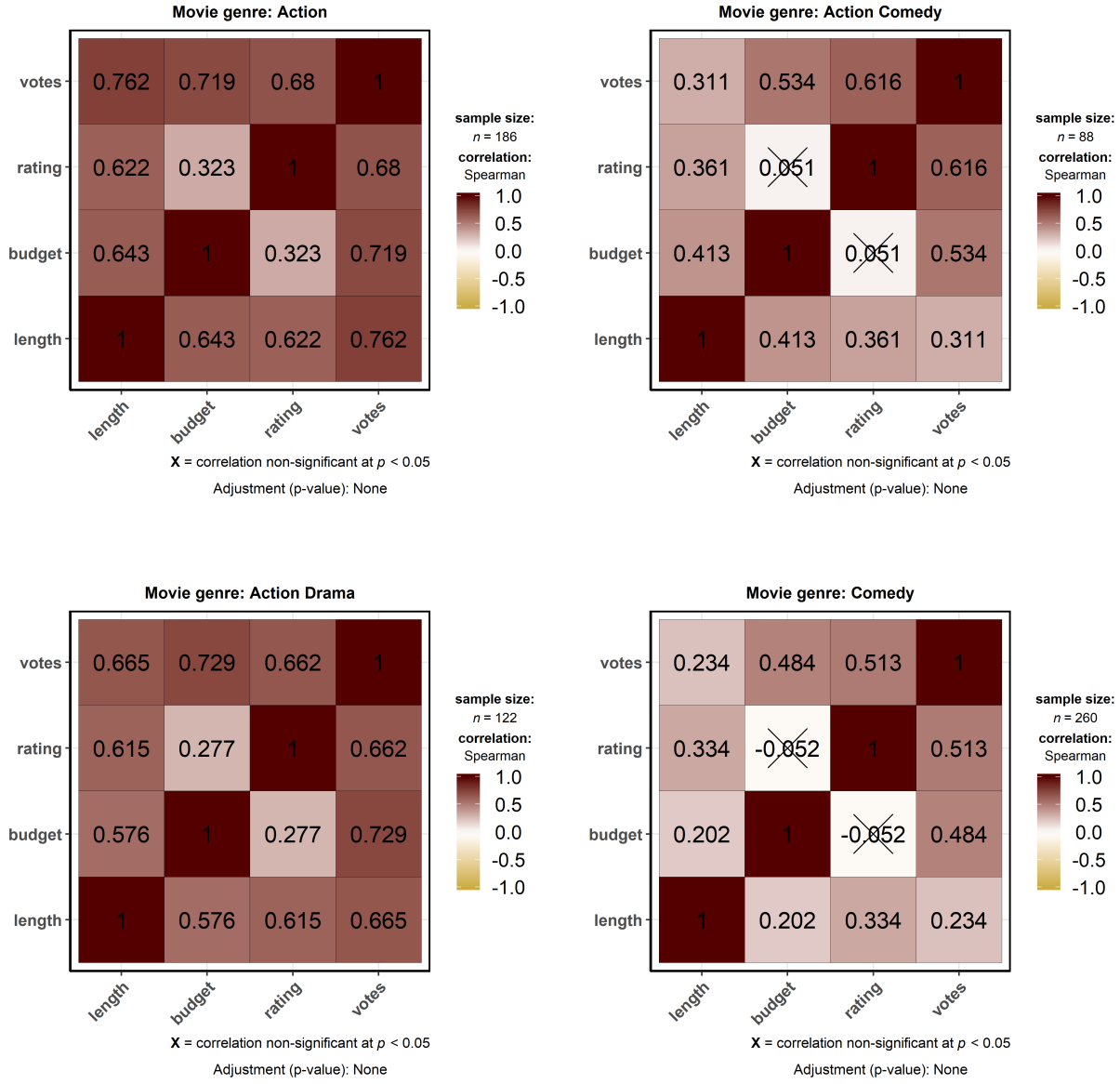
```
# for reproducibility
set.seed(123)

# as a default this function outputs a correlalogram plot
ggstatsplot::ggcorrmat(
  data = ggplot2::msleep,
  corr.method = "robust", # correlation method
  sig.level = 0.001, # threshold of significance
  p.adjust.method = "holm", # p-value adjustment method for multiple comparisons
  cor.vars = c(sleep_rem, awake:bodywt), # a range of variables can be selected
  cor.vars.names = c(
    "REM sleep", # variable names
    "time awake",
    "brain weight",
    "body weight"
  ),
  matrix.type = "upper", # type of visualization matrix
  colors = c("#B2182B", "white", "#4D4D4D"),
  title = "Correlalogram for mammals sleep dataset",
  subtitle = "sleep units: hours; weight units: kilograms"
)
```



Note that if there are `NA`s present in the selected variables, the legend will display minimum, median, and maximum number of pairs used for correlation tests.

There is a `grouped_` variant of this function that makes it easy to repeat the same operation across a **single** grouping variable:

```
# for reproducibility
set.seed(123)

# plot
# let's use only 50% of the data to speed up the process
ggstatsplot::grouped_ggcorrmat(
  data = dplyr::filter(
    .data = ggstatsplot::movies_long,
    genre %in% c("Action", "Action Comedy", "Action Drama", "Comedy")
  ),
  cor.vars = length:votes,
  corr.method = "np",
  colors = c("#cbac43", "white", "#550000"),
  grouping.var = genre, # grouping variable
  digits = 3, # number of digits after decimal point
  title.prefix = "Movie genre",
  messages = FALSE,
  nrow = 2
)
```



5.8.1 Summary of tests

Following tests are carried out for each type of analyses. Additionally, the correlation coefficients (and their confidence intervals) are used as effect sizes-

Type	Test	CI?
Parametric	Pearson's correlation coefficient	Yes
Non-parametric	Spearman's rank correlation coefficient	Yes
Robust	Percentage bend correlation coefficient	No
Bayes Factor	Pearson's correlation coefficient	No

For examples and more information, see the `ggcorrmat` vignette: https://indrajeetpatil.github.io/

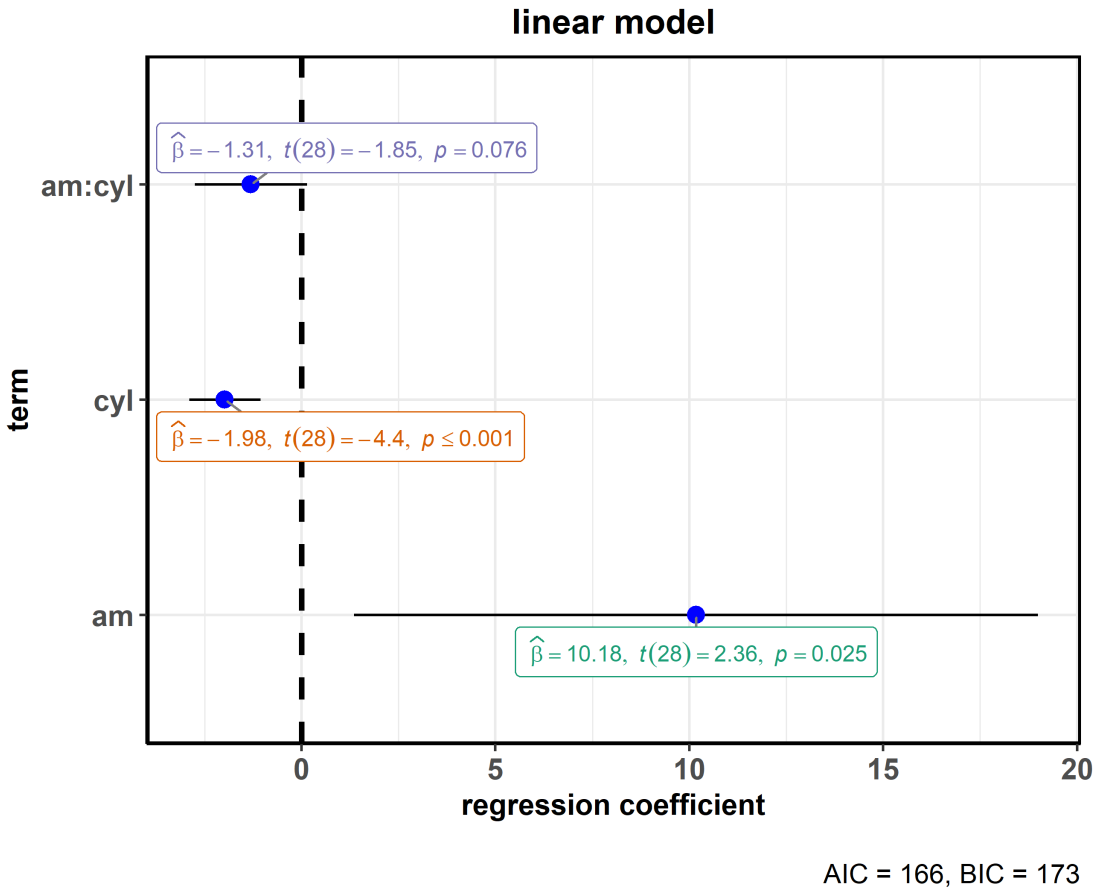
ggstatsplot/articles/web_only/ggcorrmat.html

5.9 ggcoefstats

`ggcoefstats` creates a dot-and-whisker plot for regression models. Although the statistical models displayed in the plot may differ based on the class of models being investigated, there are few aspects of the plot that will be invariant across models:

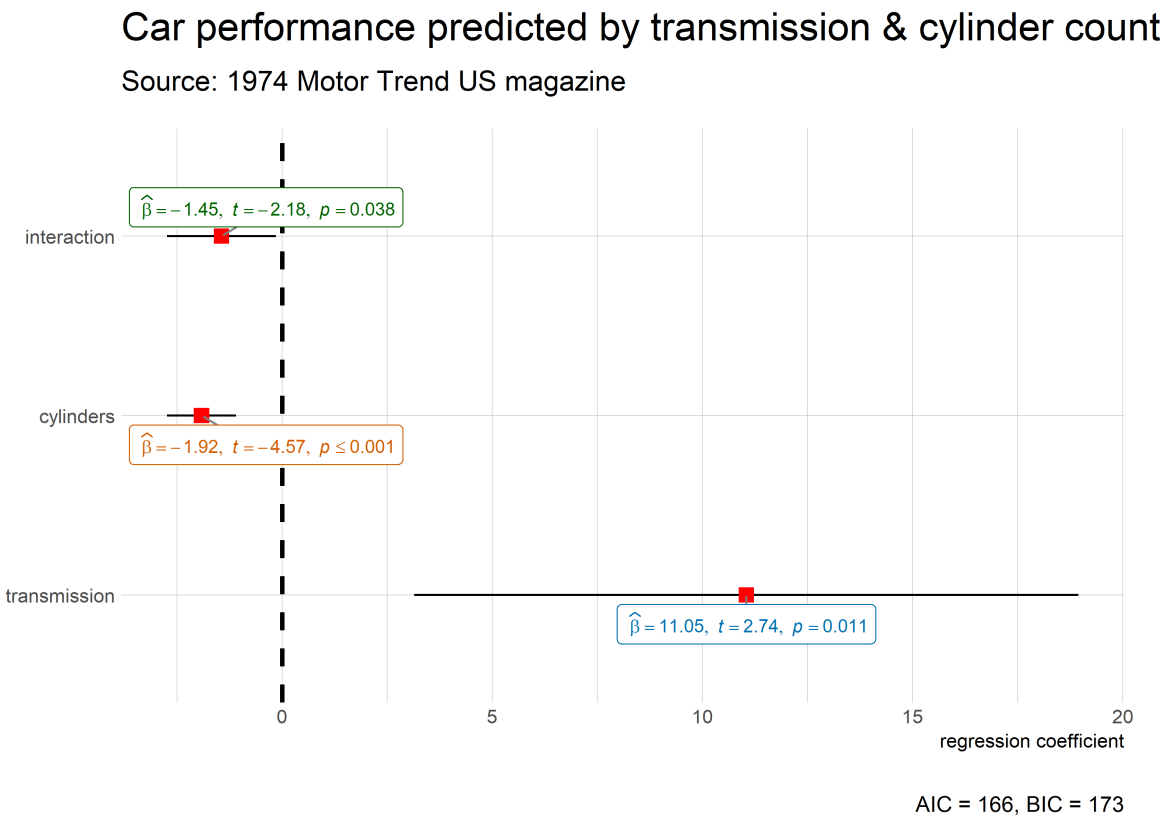
- The dot-whisker plot contains a dot representing the **estimate** and their **confidence intervals** (`95%` is the default). The estimate can either be effect sizes (for tests that depend on the `F` statistic) or regression coefficients (for tests with `t` and `z` statistic), etc. The function will, by default, display a helpful `x`-axis label that should clear up what estimates are being displayed. The confidence intervals can sometimes be asymmetric if bootstrapping was used.
- The caption will always contain diagnostic information, if available, about models that can be useful for model selection: The smaller the Akaike's Information Criterion (**AIC**) and the Bayesian Information Criterion (**BIC**) values, the "better" the model is. Additionally, the higher the **log-likelihood** value the "better" is the model fit.
- The output of this function will be a `ggplot2` object and, thus, it can be further modified (e.g., change themes, etc.) with `ggplot2` functions.

```
# for reproducibility
set.seed(123)

# model
mod <- stats::lm(formula = mpg ~ am * cyl, data = mtcars)

# plot
ggstatsplot::ggcoefstats(x = mod, title = "linear model")
```



This default plot can be further modified to one's liking with additional arguments (also, let's use a robust linear model instead of a simple linear model now):

```
# for reproducibility
set.seed(123)

# model
mod <- MASS::rlm(formula = mpg ~ am * cyl, data = mtcars)

# plot
ggstatsplot::ggcoefstats(
  x = mod,
  point.color = "red",
  point.shape = 15,
  stats.label.color = c("#0072B2", "#D55E00", "darkgreen"),
  title = "Car performance predicted by transmission & cylinder count",
  subtitle = "Source: 1974 Motor Trend US magazine",
  ggtheme = hrbrthemes::theme_ipsum_ps(),
  ggstatsplot.layer = FALSE
) + # further modification with the ggplot2 commands
  ggplot2::scale_y_discrete(labels = c("transmission", "cylinders", "interaction")) +
  ggplot2::labs(x = "regression coefficient", y = NULL)
```



Most of the regression models that are supported in the `broom` and `broom.mixed` packages with `tidy` and `glance` methods are also supported by `ggcoefstats`. For example-

`aareg`, `anova`, `aov`, `aovlist`, `Arima`, `bglmerMod`, `bigglm`, `biglm`, `blavaan`, `bmlm`, `blmerMod`, `bracl`, `brglm2`, `brmsfit`, `btergm`, `cch`, `cgam`, `cgamm`, `clm`, `clmm`, `coeftest`, `confusionMatrix`, `coxph`, `cpglm`, `cpglmm`, `complmrob`, `drc`, `emmGrid`, `epi.2by2`, `ergm`, `feis`, `felm`, `fitdistr`, `flexsurvreg`, `glmc`, `glmerMod`, `glmmTMB`, `gls`, `gam`, `Gam`, `gamlss`, `garch`, `glm`, `glmmadmb`, `glmmPQL`, `glmRob`, `glmrob`, `glmx`, `gmm`, `hurdle`, `ivreg`, `iv_robust`, `lavaan`, `lm`, `lm.beta`, `lmerMod`, `lmodel2`, `lmRob`, `lmrob`, `mcmc`, `MCMCglmm`, `mclogit`, `mmclogit`, `mediate`, `mixor`, `mjoint`, `mle2`, `mlm`, `multinom`, `negbin`, `nlmerMod`, `nlrq`, `nlreg`, `nls`, `orcutt`, `plm`, `polr`, `ridgelm`, `rjags`, `rlm`, `rlmerMod`, `rq`, `rqss`, `slm`, `speedglm`, `speedlm`, `stanfit`, `stanreg`, `survreg`, `svyglm`, `svyolr`, `svyglm`, `tobit`, `truncreg`, `vgam`, `wbgee`, `wblm`, etc.

Although not shown here, this function can also be used to carry out both frequentist and Bayesian random-effects meta-analysis.

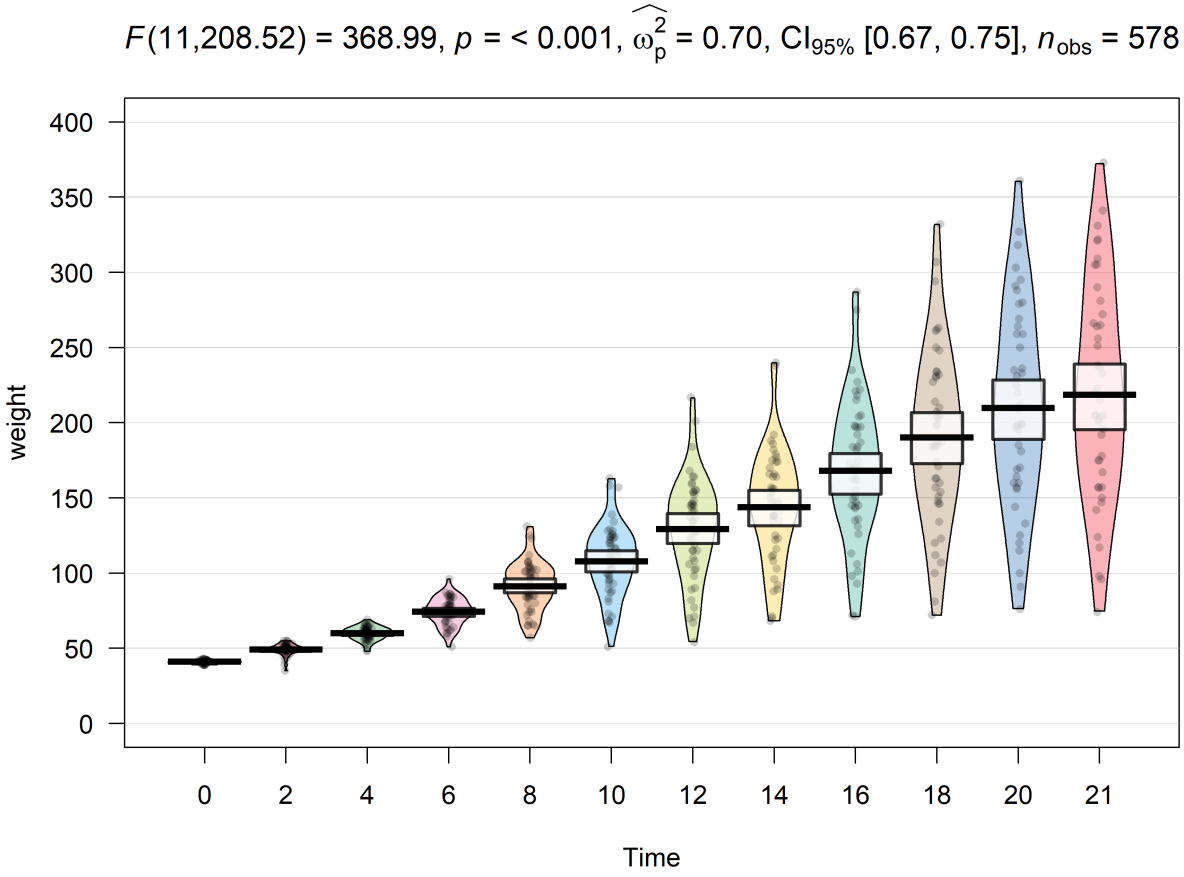
For a more exhaustive account of this function, see the associated vignette- https://indrajeetpatil.github.io/ggstatsplot/articles/web_only/ggcoefstats.html

5.10 Working with custom plots

Sometimes you may not like the defaults in a plot produced by `ggstatsplot`. In such cases, you can use other custom plots (from `ggplot2` or other graphics packages in `R`) and still use `ggstatsplot` functions to display results from relevant statistical test.

For example, in the following chunk, we will create plot (*pirateplot*) using `yarrr` package and use `ggstatsplot` function for extracting results.

```r
# for reproducibility
set.seed(123)

# loading the needed libraries
library(yarrr)
library(ggstatsplot)

# using `ggstatsplot` to get call with statistical results
stats_results <-
  ggstatsplot::ggbetweenstats(
    data = ChickWeight,
    x = Time,
    y = weight,
    return = "subtitle",
    messages = FALSE
  )

# using `yarrr` to create plot
yarrr::pirateplot(
  formula = weight ~ Time,
  data = ChickWeight,
  theme = 1,
  main = stats_results
)
```



5.11 Overall consistency in API

Attempt has been made to make the application program interface (API) consistent enough that no struggle is expected while thinking about specifying function calls-

- When a given function depends on variables in a dataframe, `data` argument must always be specified.
- Often, package functions relevant for between-subjects versus within-subjects design expect tidy/long versus Cartesian/wide format data, respectively. `ggstatsplot` functions consistently expect tidy/long form data.
- All functions accept both quoted (`x = "var1"`) and unquoted (`x = var1`) arguments.

These set principles combined with the fact that almost all functions produce publication-ready plots that require very few arguments if one finds the aesthetic and statistical defaults satisfying make the syntax much less cognitively demanding and easy to remember/reconstruct.

5.12 Helpful messages and aesthetic modifications

`ggstatsplot` is a very chatty package and will by default print helpful notes on assumptions about statistical tests, warnings, etc. If you don't want your console to be cluttered with such messages, they can be turned off by setting argument `messages = FALSE` in the function call.

```r
# group difference tests - between-subjects
ggbetweenstats(data, x, y, ...)

# group difference tests - within-subjects
ggwithinstats(data, x, y, ...)

# correlation analyses
ggscatterstats(data, x, y, ...)

# correlation matrix
ggcorrmat(data, ...)

# numeric variable
gghistostats(data, x, ...)

# labelled numeric variable
ggdotplotstats(data, x, y, ...)

# contingency table analyses
ggpiestats(data, x, y, ...)
ggbarstats(data, x, y, ...)

# goodness of fit (proportion) tests
ggpiestats(data, x, ...)

# regression analyses
ggcoefstats(x, ...)
```

Figure 8: Consistent API

All relevant functions in `ggstatsplot` have a `return` argument which can be used to not only return plots (which is the default), but also to return a `subtitle` or `caption`, which are objects of type `call` and can be used to display statistical details in conjunction with a custom plot and at a custom location in the plot.

Additionally, all functions share the `ggtheme` and `palette` arguments that can be used to specify your favorite `ggplot` theme and color palette.

6 Acknowledgments

The authors would like to thank Chuck Powell, Daniel Heck, and Will Beasley for their contributions to the package. The authors would also like to acknowledge the help and support provided by the larger `#rstats` community on Twitter and StackOverflow for the development of this package. We also appreciate helpful discussions with Fiery Cushman.

7 Appendix

7.1 Appendix A: Documentation

There are three main documents one can rely on to learn how to use `ggstatsplot`:

- **Presentation**: The quickest (and the most fun) way to get an overview of the philosophy behind this package and the offered functionality is to go through the following slides: https://indrajeetpatil.github.io/ggstatsplot_slides/slides/ggstatsplot_presentation.html#1
- **Manual**:
  The `CRAN` reference manual provides detailed documentation about arguments for each function and examples: https://cran.r-project.org/web/packages/ggstatsplot/ggstatsplot.pdf
- **Vignettes**:
  Vignettes contain probably the most detailed exposition. Every single function in `ggstatsplot` has an associated vignette which describes in depth how to use the function and modify the defaults to customize the plot to your liking. All these vignettes can be accessed from the package website: https://indrajeetpatil.github.io/ggstatsplot/articles/

7.2 Appendix B: Suggestions

If you find any bugs or have any suggestions/remarks, please file an issue on `GitHub` repository for this package: https://github.com/IndrajeetPatil/ggstatsplot/issues

7.3 Appendix C: Contributing

`ggstatsplot` is happy to receive bug reports, suggestions, questions, and (most of all) contributions to fix problems and add features. Pull Requests for contributions are encouraged.

Here are some simple ways in which one can contribute (in the increasing order of commitment):

- Read and correct any inconsistencies in the documentation
- Raise issues about bugs or wanted features
- Review code
- Add new functionality (in the form of new plotting functions or helpers for preparing subtitles)

Please note that this project is released with a Contributor Code of Conduct. By participating in this project you agree to abide by its terms.

7.4 Appendix D: Session information

For reproducibility purposes, the details about the session information in which this document was rendered, see- https://indrajeetpatil.github.io/ggstatsplot/articles/web_only/session_info.html

References

Association, A. P. (2009). *Publication Manual of the American Psychological Association, 6th Edition.* Washington, DC: American Psychological Association.

Cleveland, W. S. (1985). *The Elements of Graphing Data* (1st edition). Monterey, Cal: Wadsworth, Inc.

Lilienfeld, S. O., Sauvigné, K. C., Lynn, S. J., Cautin, R. L., Latzman, R. D., & Waldman, I. D. (2015). Fifty psychological and psychiatric terms to avoid: A list of inaccurate, misleading, misused, ambiguous, and logically confused words and phrases. *Frontiers in Psychology*, *6*. https://doi.org/10.3389/fpsyg.2015.01100

Nuijten, M. B., Hartgerink, C. H. J., van Assen, M. A. L. M., Epskamp, S., & Wicherts, J. M. (2016). The prevalence of statistical reporting errors in psychology (19852013). *Behavior Research Methods*, *48*(4), 1205–1226. https://doi.org/10.3758/s13428-015-0664-2

Tufte, E. R. (2001). *The Visual Display of Quantitative Information* (2nd edition edition). Cheshire, Conn: Graphics Press.

Wickham, H. (2014). Tidy Data. *Journal of Statistical Software*, *59*(1), 1–23. https://doi.org/10.18637/jss.v059.i10

Wickham, H. (2016). *Ggplot2: Elegant Graphics for Data Analysis* (2nd ed. 2016 edition). New York, NY: Springer.