

# Package: cardx (via r-universe)

January 15, 2025

**Title** Extra Analysis Results Data Utilities

**Version** 0.2.2.9012

**Description** Create extra Analysis Results Data (ARD) summary objects.

The package supplements the simple ARD functions from the 'cards' package, exporting functions to put statistical results in the ARD format. These objects are used and re-used to construct summary tables, visualizations, and written reports.

**License** Apache License 2.0

**URL** <https://insightsengineering.github.io/cardx/>,  
<https://github.com/insightsengineering/cardx/>

**BugReports** <https://github.com/insightsengineering/cardx/issues>

**Depends** R (>= 4.1)

**Imports** cards (>= 0.4.0), cli (>= 3.6.1), dplyr (>= 1.1.2), glue (>= 1.6.2), lifecycle (>= 1.0.3), rlang (>= 1.1.1), tidyr (>= 1.3.0)

**Suggests** aod (>= 1.3.3), broom (>= 1.0.5), broom.helpers (>= 1.17.0), broom.mixed (>= 0.2.9), car (>= 3.1-2), effectsize (>= 0.8.8), emmeans (>= 1.7.3), geopack (>= 1.3.2), ggsvrfit (>= 1.1.0), lme4 (>= 1.1-35.3), parameters (>= 0.20.2), smd (>= 0.6.6), spelling (>= 2.3.0), survey (>= 4.2), survival (>= 3.6-4), testthat (>= 3.2.0), withr (>= 2.5.0)

**Config/Needs/website** insightsengineering/nesttemplate

**Config/testthat/edition** 3

**Config/testthat/parallel** true

**Encoding** UTF-8

**Language** en-US

**LazyData** true

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.2

**Config/pak/sysreqs** libicu-dev

**Repository** <https://ddsjoberg.r-universe.dev>

**RemoteUrl** <https://github.com/insightsengineering/cardx>

**RemoteRef** HEAD

**RemoteSha** 07ebffe9577b3b0733791622e966e89f57e9a2d7

## Contents

ard_aod_wald_test . . . . .	3
ard_attributes.survey.design . . . . .	4
ard_car_anova . . . . .	5
ard_car_vif . . . . .	5
ard_categorical.survey.design . . . . .	6
ard_categorical_ci . . . . .	7
ard_categorical_ci.survey.design . . . . .	9
ard_categorical_max . . . . .	10
ard_continuous.survey.design . . . . .	11
ard_continuous_ci . . . . .	13
ard_continuous_ci.survey.design . . . . .	14
ard_dichotomous.survey.design . . . . .	15
ard_effectsize_cohens_d . . . . .	16
ard_effectsize_hedges_g . . . . .	18
ard_emmeans_mean_difference . . . . .	19
ard_missing.survey.design . . . . .	21
ard_regression . . . . .	22
ard_regression_basic . . . . .	23
ard_smd_smd . . . . .	24
ard_stats_anova . . . . .	25
ard_stats_aov . . . . .	27
ard_stats_chisq_test . . . . .	27
ard_stats_fisher_test . . . . .	28
ard_stats_kruskal_test . . . . .	29
ard_stats_mcnemar_test . . . . .	30
ard_stats_mood_test . . . . .	31
ard_stats_oneway_test . . . . .	32
ard_stats_poisson_test . . . . .	32
ard_stats_prop_test . . . . .	34
ard_stats_t_test . . . . .	34
ard_stats_t_test_onesample . . . . .	36
ard_stats_wilcox_test . . . . .	37
ard_stats_wilcox_test_onesample . . . . .	38
ard_survey_svychisq . . . . .	39
ard_survey_svyranktest . . . . .	40
ard_survey_svytttest . . . . .	41
ard_survival_survdiff . . . . .	41
ard_survival_survfit . . . . .	42
ard_survival_survfit_diff . . . . .	45
ard_total_n.survey.design . . . . .	45

<code>ard_aod_wald_test</code>	3
<code>construction_helpers</code> . . . . .	46
<code>proportion_ci</code> . . . . .	48
<b>Index</b>	<b>52</b>

---

<code>ard_aod_wald_test</code>	<i>ARD Wald Test</i>
--------------------------------	----------------------

---

## Description

Function takes a regression model object and calculates Wald statistical test using `aod::wald.test()`.

## Usage

```
ard_aod_wald_test(
  x,
  tidy_fun = broom.helpers::tidy_with_broom_or_parameters,
  ...
)
```

## Arguments

<code>x</code>	regression model object
<code>tidy_fun</code>	(function) a tidier. Default is <code>broom.helpers::tidy_with_broom_or_parameters</code>
<code>...</code>	arguments passed to <code>aod::wald.test(...)</code>

## Value

data frame

## Examples

```
lm(AGE ~ ARM, data = cards::ADSL) |>
  ard_aod_wald_test()
```

---

```
ard_attributes.survey.design
  ARD Attributes
```

---

## Description

Add variable attributes to an ARD data frame.

- The label attribute will be added for all columns, and when no label is specified and no label has been set for a column using the `label=` argument, the column name will be placed in the label statistic.
- The class attribute will also be returned for all columns.
- Any other attribute returned by `attributes()` will also be added, e.g. factor levels.

## Usage

```
## S3 method for class 'survey.design'
ard_attributes(data, variables = everything(), label = NULL, ...)
```

## Arguments

<code>data</code>	( <code>survey.design</code> ) a design object often created with <code>survey::svydesign()</code> .
<code>variables</code>	( <code>tidy-select</code> ) variables to include
<code>label</code>	(named list) named list of variable labels, e.g. <code>list(cyl = "No. Cylinders")</code> . Default is <code>NULL</code>
<code>...</code>	These dots are for future extensions and must be empty.

## Value

an ARD data frame of class 'card'

## Examples

```
data(api, package = "survey")
dclus1 <- survey::svydesign(id = ~dnum, weights = ~pw, data = apiclus1, fpc = ~fpc)

ard_attributes(
  data = dclus1,
  variables = c(sname, dname),
  label = list(sname = "School Name", dname = "District Name")
)
```

---

ard_car_anova	<i>ARD ANOVA from car Package</i>
---------------	-----------------------------------

---

**Description**

Function takes a regression model object and calculated ANOVA using `car::Anova()`.

**Usage**

```
ard_car_anova(x, ...)
```

**Arguments**

x	regression model object
...	arguments passed to <code>car::Anova(...)</code>

**Value**

data frame

**Examples**

```
lm(AGE ~ ARM, data = cards::ADSL) |>
  ard_car_anova()

glm(vs ~ factor(cyl) + factor(am), data = mtcars, family = binomial) |>
  ard_car_anova(test.statistic = "Wald")
```

---

ard_car_vif	<i>Regression VIF ARD</i>
-------------	---------------------------

---

**Description**

Function takes a regression model object and returns the variance inflation factor (VIF) using `car::vif()` and converts it to a ARD structure

**Usage**

```
ard_car_vif(x, ...)
```

**Arguments**

x	regression model object See <code>car::vif()</code> for details
...	arguments passed to <code>car::vif(...)</code>

**Value**

data frame

**Examples**

```
lm(AGE ~ ARM + SEX, data = cards::ADSL) |>
  ard_car_vif()
```

---

ard\_categorical.survey.design

*ARD Categorical Survey Statistics*

---

**Description**

Compute tabulations on survey-weighted data.

The counts and proportion ("N", "n", "p") are calculated using `survey::svytable()`, and the standard errors and design effect ("p.std.error", "deff") are calculated using `survey::svymean()`.

The unweighted statistics are calculated with `cards::ard_categorical.data.frame()`.

**Usage**

```
## S3 method for class 'survey.design'
ard_categorical(
  data,
  variables,
  by = NULL,
  statistic = everything() ~ c("n", "N", "p", "p.std.error", "deff", "n_unweighted",
    "N_unweighted", "p_unweighted"),
  denominator = c("column", "row", "cell"),
  fmt_fn = NULL,
  stat_label = everything() ~ list(p = "%", p.std.error = "SE%", deff =
    "Design Effect", n_unweighted = "Unweighted n", N_unweighted = "Unweighted N",
    p_unweighted = "Unweighted %"),
  ...
)
```

**Arguments**

data	(survey.design) a design object often created with <code>survey::svydesign()</code> .
variables	(tidy-select) columns to include in summaries.
by	(tidy-select) results are calculated for <b>all combinations</b> of the column specified and the variables. A single column may be specified.

statistic	( <a href="#">formula-list-selector</a> ) a named list, a list of formulas, or a single formula where the list element is a character vector of statistic names to include. See default value for options.
denominator	(string) a string indicating the type proportions to calculate. Must be one of "column" (the default), "row", and "cell".
fmt_fn	( <a href="#">formula-list-selector</a> ) a named list, a list of formulas, or a single formula where the list element is a named list of functions (or the RHS of a formula), e.g. <code>list(mpg = list(mean = \(x) round(x, digits</code>
stat_label	( <a href="#">formula-list-selector</a> ) a named list, a list of formulas, or a single formula where the list element is either a named list or a list of formulas defining the statistic labels, e.g. <code>everything() ~ list(mean = "Mean", sd = "SD")</code> or <code>everything() ~ list(mean ~ "Mean", sd ~ "SD")</code> .
...	These dots are for future extensions and must be empty.

**Value**

an ARD data frame of class 'card'

**Examples**

```
svy_titanic <- survey::svydesign(~1, data = as.data.frame(Titanic), weights = ~Freq)
ard_categorical(svy_titanic, variables = c(Class, Age), by = Survived)
```

---

ard\_categorical\_ci      *ARD Proportion Confidence Intervals*

---

**Description****[Experimental]**

Calculate confidence intervals for proportions.

**Usage**

```
ard_categorical_ci(data, ...)

## S3 method for class 'data.frame'
ard_categorical_ci(
  data,
  variables,
  by = dplyr::group_vars(data),
  method = c("waldcc", "wald", "clopper-pearson", "wilson", "wilsoncc", "strat_wilson",
            "strat_wilsoncc", "agresti-coull", "jeffreys"),
```

```

denominator = c("column", "row", "cell"),
conf.level = 0.95,
value = list(where(is_binary) ~ 1L, where(is.logical) ~ TRUE),
strata = NULL,
weights = NULL,
max.iterations = 10,
...
)

```

## Arguments

data	(data.frame) a data frame
...	Arguments passed to methods.
variables	( <a href="#">tidy-select</a> ) columns to include in summaries. Columns must be class <logical> or <numeric> values coded as c(0, 1).
by	( <a href="#">tidy-select</a> ) columns to stratify calculations by.
method	(string) string indicating the type of confidence interval to calculate. Must be one of . See ?proportion_ci for details.
denominator	(string) Must be one of 'column' (default), 'row', and 'cell', which specifies the direction of the calculation/denominator. Argument is similar to cards::ard_categorical(denominator)
conf.level	(numeric) a scalar in (0, 1) indicating the confidence level. Default is 0.95
value	( <a href="#">formula-list-selector</a> ) function will calculate the CIs for all levels of the variables specified. Use this argument to instead request only a single level by summarized. Default is list(where(is_binary) ~ 1L, where(is.logical) ~ TRUE), where columns coded as 0/1 and TRUE/FALSE will summarize the 1 and TRUE levels.
strata, weights, max.iterations	arguments passed to proportion_ci_strat_wilson(), when method='strat_wilson'

## Value

an ARD data frame

## Examples

```

# compute CI for binary variables
ard_categorical_ci(mtcars, variables = c(vs, am), method = "wilson")

# compute CIs for each level of a categorical variable
ard_categorical_ci(mtcars, variables = cyl, method = "jeffreys")

```



---

```
ard_categorical_ci.survey.design
  ARD survey categorical CIs
```

---

## Description

Confidence intervals for categorical variables calculated via `survey::svyciprop()`.

## Usage

```
## S3 method for class 'survey.design'
ard_categorical_ci(
  data,
  variables,
  by = NULL,
  method = c("logit", "likelihood", "asin", "beta", "mean", "xlogit"),
  conf.level = 0.95,
  value = list(where(is_binary) ~ 1L, where(is.logical) ~ TRUE),
  df = survey::degf(data),
  ...
)
```

## Arguments

<code>data</code>	( <code>survey.design</code> ) a design object often created with <code>survey::svydesign()</code> .
<code>variables</code>	( <code>tidy-select</code> ) columns to include in summaries.
<code>by</code>	( <code>tidy-select</code> ) results are calculated for <b>all combinations</b> of the columns specified, including unobserved combinations and unobserved factor levels.
<code>method</code>	(string) Method passed to <code>survey::svyciprop(method)</code>
<code>conf.level</code>	(numeric) a scalar in (0, 1) indicating the confidence level. Default is 0.95
<code>value</code>	( <code>formula-list-selector</code> ) function will calculate the CIs for all levels of the variables specified. Use this argument to instead request only a single level by summarized. Default is <code>list(where(is_binary) ~ 1L, where(is.logical) ~ TRUE)</code> , where columns coded as 0/1 and TRUE/FALSE will summarize the 1 and TRUE levels.
<code>df</code>	(numeric) denominator degrees of freedom, passed to <code>survey::svyciprop(df)</code> . Default is <code>survey::degf(data)</code> .
<code>...</code>	arguments passed to <code>survey::svyciprop()</code>

**Value**

ARD data frame

**Examples**

```
data(api, package = "survey")
dclus1 <- survey::svydesign(id = ~dnum, weights = ~pw, data = apiclus1, fpc = ~fpc)

ard_categorical_ci(dclus1, variables = sch.wide)
ard_categorical_ci(dclus1, variables = sch.wide, value = sch.wide ~ "Yes", method = "xlogit")
```

---

ard_categorical_max	<i>ARD to Calculate Categorical Occurrence Rates by Maximum Level Per Unique ID</i>
---------------------	---

---

**Description**

Function calculates categorical variable level occurrences rates by maximum level per unique ID. Each variable in `variables` is evaluated independently and then results for all variables are stacked. Only the highest-ordered level will be counted for each unique ID. Unordered, non-numeric variables will be converted to factor and the default level order used for ordering.

**Usage**

```
ard_categorical_max(
  data,
  variables,
  id,
  by = dplyr::group_vars(data),
  statistic = everything() ~ c("n", "p", "N"),
  denominator = NULL,
  fmt_fn = NULL,
  stat_label = everything() ~ cards::default_stat_labels(),
  quiet = FALSE,
  ...
)
```

**Arguments**

data	(data.frame) a data frame
variables	( <a href="#">tidy-select</a> ) The categorical variables for which occurrence rates per unique ID (by maximum level) will be calculated.

id	( <a href="#">tidy-select</a> ) Argument used to subset data to identify rows in data to calculate categorical variable level occurrence rates.
by	( <a href="#">tidy-select</a> ) columns to tabulate by in the series of ARD function calls. Any rows with NA or NaN values are removed from all calculations.
statistic	( <a href="#">formula-list-selector</a> ) a named list, a list of formulas, or a single formula where the list element one or more of c("n", "N", "p") (or the RHS of a formula).
denominator	(data.frame, integer) An optional argument to change the denominator used for "N" and "p" statistic calculations. Defaults to NULL, in which case <code>dplyr::distinct(data, dplyr::pick(all_of(c(id, by))))</code> is used for these calculations. See <a href="#">cards::ard_categorical()</a> for more details on specifying denominators.
fmt_fn	( <a href="#">formula-list-selector</a> ) a named list, a list of formulas, or a single formula where the list element is a named list of functions (or the RHS of a formula), e.g. <code>list(mpg = list(mean = \(x) round(x, digits</code>
stat_label	( <a href="#">formula-list-selector</a> ) a named list, a list of formulas, or a single formula where the list element is either a named list or a list of formulas defining the statistic labels, e.g. <code>everything() ~ list(n = "n", p = "pct")</code> or <code>everything() ~ list(n ~ "n", p ~ "pct")</code> .
quiet	(scalar logical) Logical indicating whether to suppress additional messaging. Default is FALSE.
...	Arguments passed to methods.

**Value**

an ARD data frame of class 'card'

**Examples**

```
# Occurrence Rates by Max Level (Highest Severity) -----
ard_categorical_max(
  cards::ADAE,
  variables = c(AESER, AESEV),
  id = USBJID,
  by = TRTA,
  denominator = cards::ADSL |> dplyr::rename(TRTA = ARM)
)
```

**Description**

Returns an ARD of weighted statistics using the {survey} package.

**Usage**

```
## S3 method for class 'survey.design'
ard_continuous(
  data,
  variables,
  by = NULL,
  statistic = everything() ~ c("median", "p25", "p75"),
  fmt_fn = NULL,
  stat_label = NULL,
  ...
)
```

**Arguments**

data	(survey.design) a design object often created with <code>survey::svydesign()</code> .
variables	(tidy-select) columns to include in summaries.
by	(tidy-select) results are calculated for <b>all combinations</b> of the columns specified, including unobserved combinations and unobserved factor levels.
statistic	(formula-list-selector) a named list, a list of formulas, or a single formula where the list element is a character vector of statistic names to include. See below for options.
fmt_fn	(formula-list-selector) a named list, a list of formulas, or a single formula where the list element is a named list of functions (or the RHS of a formula), e.g. <code>list(mpg = list(mean = \(x) round(x, digits</code>
stat_label	(formula-list-selector) a named list, a list of formulas, or a single formula where the list element is either a named list or a list of formulas defining the statistic labels, e.g. <code>everything() ~ list(mean = "Mean", sd = "SD")</code> or <code>everything() ~ list(mean ~ "Mean", sd ~ "SD")</code> .
...	These dots are for future extensions and must be empty.

**Value**

an ARD data frame of class 'card'

**statistic argument**

The following statistics are available: 'mean', 'median', 'min', 'max', 'sum', 'var', 'sd', 'mean.std.error', 'deff', 'p##', where 'p##' is the percentiles and ## is an integer between 0 and 100.

**Examples**

```

data(api, package = "survey")
dclus1 <- survey::svydesign(id = ~dnum, weights = ~pw, data = apiclus1, fpc = ~fpc)

ard_continuous(
  data = dclus1,
  variables = api00,
  by = stype
)

```

---

ard\_continuous\_ci      *ARD continuous CIs*

---

**Description**

One-sample confidence intervals for continuous variable means and medians.

**Usage**

```

ard_continuous_ci(data, ...)

## S3 method for class 'data.frame'
ard_continuous_ci(
  data,
  variables,
  by = dplyr::group_vars(data),
  conf.level = 0.95,
  method = c("t.test", "wilcox.test"),
  ...
)

```

**Arguments**

data	(data.frame) a data frame. See below for details.
...	arguments passed to <code>t.test()</code> or <code>wilcox.test()</code>
variables	( <a href="#">tidy-select</a> ) column names to be compared. Independent t-tests will be computed for each variable.
by	( <a href="#">tidy-select</a> ) optional column name to compare by.
conf.level	(scalar numeric) confidence level for confidence interval. Default is 0.95.
method	(string) a string indicating the method to use for the confidence interval calculation. Must be one of "t.test" or "wilcox.test"

**Value**

ARD data frame

**Examples**

```
ard_continuous_ci(mtcars, variables = c(mpg, hp), method = "wilcox.test")
ard_continuous_ci(mtcars, variables = mpg, by = am, method = "t.test")
```

---

```
ard_continuous_ci.survey.design
      ARD survey continuous CIs
```

---

**Description**

One-sample confidence intervals for continuous variables' means and medians. Confidence limits are calculated with `survey::svymean()` and `survey::svyquantile()`.

**Usage**

```
## S3 method for class 'survey.design'
ard_continuous_ci(
  data,
  variables,
  by = NULL,
  method = c("svymean", "svymedian.mean", "svymedian.beta", "svymedian.xlogit",
             "svymedian.asin", "svymedian.score"),
  conf.level = 0.95,
  df = survey::degf(data),
  ...
)
```

**Arguments**

<code>data</code>	( <code>survey.design</code> ) a design object often created with <code>survey::svydesign()</code> .
<code>variables</code>	( <code>tidy-select</code> ) columns to include in summaries.
<code>by</code>	( <code>tidy-select</code> ) results are calculated for <b>all combinations</b> of the columns specified, including unobserved combinations and unobserved factor levels.
<code>method</code>	(string) Method for confidence interval calculation. When "svymean", the calculation is computed via <code>survey::svymean()</code> . Otherwise, it is calculated via <code>survey::svyquantile(interval.type)</code> .
<code>conf.level</code>	(scalar numeric) confidence level for confidence interval. Default is 0.95.

```

df          (numeric)
            denominator degrees of freedom, passed to survey::confint(df). Default is
            survey::degf(data).

...        arguments passed to survey::confint()

```

**Value**

ARD data frame

**Examples**

```

data(api, package = "survey")
dclus1 <- survey::svydesign(id = ~dnum, weights = ~pw, data = apiclus1, fpc = ~fpc)

ard_continuous_ci(dclus1, variables = api00)
ard_continuous_ci(dclus1, variables = api00, method = "svymedian.xlogit")

```

---

ard\_dichotomous.survey.design

*ARD Dichotomous Survey Statistics*

---

**Description**

Compute Analysis Results Data (ARD) for dichotomous summary statistics.

**Usage**

```

## S3 method for class 'survey.design'
ard_dichotomous(
  data,
  variables,
  by = NULL,
  value = cards::maximum_variable_value(data$variables[variables]),
  statistic = everything() ~ c("n", "N", "p", "p.std.error", "deff", "n_unweighted",
    "N_unweighted", "p_unweighted"),
  denominator = c("column", "row", "cell"),
  fmt_fn = NULL,
  stat_label = everything() ~ list(p = "%", p.std.error = "SE(%)", deff =
    "Design Effect", n_unweighted = "Unweighted n", N_unweighted = "Unweighted N",
    p_unweighted = "Unweighted %"),
  ...
)

```

**Arguments**

data	(survey.design) a design object often created with <code>survey::svydesign()</code> .
variables	(tidy-select) columns to include in summaries.
by	(tidy-select) results are calculated for <b>all combinations</b> of the column specified and the variables. A single column may be specified.
value	(named list) named list of dichotomous values to tabulate. Default is <code>cards::maximum_variable_value(data\$variables)</code> which returns the largest/last value after a sort.
statistic	(formula-list-selector) a named list, a list of formulas, or a single formula where the list element is a character vector of statistic names to include. See default value for options.
denominator	(string) a string indicating the type proportions to calculate. Must be one of "column" (the default), "row", and "cell".
fmt_fn	(formula-list-selector) a named list, a list of formulas, or a single formula where the list element is a named list of functions (or the RHS of a formula), e.g. <code>list(mpg = list(mean = \(x) round(x, digits = 2)</code>
stat_label	(formula-list-selector) a named list, a list of formulas, or a single formula where the list element is either a named list or a list of formulas defining the statistic labels, e.g. <code>everything() ~ list(mean = "Mean", sd = "SD")</code> or <code>everything() ~ list(mean ~ "Mean", sd ~ "SD")</code> .
...	These dots are for future extensions and must be empty.

**Value**

an ARD data frame of class 'card'

**Examples**

```
survey::svydesign(ids = ~1, data = mtcars, weights = ~1) |>
  ard_dichotomous(by = vs, variables = c(cyl, am), value = list(cyl = 4))
```

---

ard\_effectsize\_cohens\_d

*ARD Cohen's D Test*

---

**Description**

Analysis results data for paired and non-paired Cohen's D Effect Size Test using `effectsize::cohen_d()`.



**Usage**

```
ard_effectsize_cohens_d(data, by, variables, conf.level = 0.95, ...)
```

```
ard_effectsize_paired_cohens_d(data, by, variables, id, conf.level = 0.95, ...)
```

**Arguments**

data	(data.frame) a data frame. See below for details.
by	(tidy-select) column name to compare by. Must be a categorical variable with exactly two levels.
variables	(tidy-select) column names to be compared. Must be a continuous variables. Independent tests will be run for each variable.
conf.level	(scalar numeric) confidence level for confidence interval. Default is 0.95.
...	arguments passed to effectsize::cohens_d(...)
id	(tidy-select) column name of the subject or participant ID

**Details**

For the `ard_effectsize_cohens_d()` function, the data is expected to be one row per subject. The data is passed as `effectsize::cohens_d(data[[variable]]~data[[by]], data, paired = FALSE, ...)`.

For the `ard_effectsize_paired_cohens_d()` function, the data is expected to be one row per subject per by level. Before the effect size is calculated, the data are reshaped to a wide format to be one row per subject. The data are then passed as `effectsize::cohens_d(x = data_wide[[<by level 1>]], y = data_wide[[<by level 2>]])`.

**Value**

ARD data frame

**Examples**

```
cards::ADSL |>
  dplyr::filter(ARM %in% c("Placebo", "Xanomeline High Dose")) |>
  ard_effectsize_cohens_d(by = ARM, variables = AGE)

# constructing a paired data set,
# where patients receive both treatments
cards::ADSL[c("ARM", "AGE")] |>
  dplyr::filter(ARM %in% c("Placebo", "Xanomeline High Dose")) |>
  dplyr::mutate(.by = ARM, USUBJID = dplyr::row_number()) |>
  dplyr::arrange(USUBJID, ARM) |>
  dplyr::group_by(USUBJID) |>
  dplyr::filter(dplyr::n() > 1) |>
```

```
ard_effectsize_paired_cohens_d(by = ARM, variables = AGE, id = USUBJID)
```

---

```
ard_effectsize_hedges_g
```

*ARD Hedge's G Test*

---

## Description

Analysis results data for paired and non-paired Hedge's G Effect Size Test using `effectsize::hedges_g()`.

## Usage

```
ard_effectsize_hedges_g(data, by, variables, conf.level = 0.95, ...)
```

```
ard_effectsize_paired_hedges_g(data, by, variables, id, conf.level = 0.95, ...)
```

## Arguments

<code>data</code>	( <code>data.frame</code> ) a data frame. See below for details.
<code>by</code>	( <code>tidy-select</code> ) column name to compare by. Must be a categorical variable with exactly two levels.
<code>variables</code>	( <code>tidy-select</code> ) column names to be compared. Must be a continuous variable. Independent tests will be run for each variable
<code>conf.level</code>	(scalar numeric) confidence level for confidence interval. Default is 0.95.
<code>...</code>	arguments passed to <code>effectsize::hedges_g(...)</code>
<code>id</code>	( <code>tidy-select</code> ) column name of the subject or participant ID

## Details

For the `ard_effectsize_hedges_g()` function, the data is expected to be one row per subject. The data is passed as `effectsize::hedges_g(data[[variable]]~data[[by]], data, paired = FALSE, ...)`.

For the `ard_effectsize_paired_hedges_g()` function, the data is expected to be one row per subject per by level. Before the effect size is calculated, the data are reshaped to a wide format to be one row per subject. The data are then passed as `effectsize::hedges_g(x = data_wide[[<by level 1>]], y = data_wide[[`

## Value

ARD data frame

**Examples**

```
cards::ADSL |>
  dplyr::filter(ARM %in% c("Placebo", "Xanomeline High Dose")) |>
  ard_effectsize_hedges_g(by = ARM, variables = AGE)

# constructing a paired data set,
# where patients receive both treatments
cards::ADSL[c("ARM", "AGE")] |>
  dplyr::filter(ARM %in% c("Placebo", "Xanomeline High Dose")) |>
  dplyr::mutate(.by = ARM, USUBJID = dplyr::row_number()) |>
  dplyr::arrange(USUBJID, ARM) |>
  dplyr::group_by(USUBJID) |>
  dplyr::filter(dplyr::n() > 1) |>
  ard_effectsize_paired_hedges_g(by = ARM, variables = AGE, id = USUBJID)
```

---

ard\_emmeans\_mean\_difference

*ARD for LS Mean Difference*


---

**Description**

This function calculates least-squares mean differences using the 'emmeans' package using the following

```
emmeans::emmeans(object = <regression model>, specs = ~ <primary covariate>) |>
  emmeans::contrast(method = "pairwise") |>
  summary(infer = TRUE, level = <confidence level>)
```

The arguments `data`, `formula`, `method`, `method.args`, `package` are used to construct the regression model via `cardx::construct_model()`.

**Usage**

```
ard_emmeans_mean_difference(
  data,
  formula,
  method,
  method.args = list(),
  package = "base",
  response_type = c("continuous", "dichotomous"),
  conf.level = 0.95,
  primary_covariate = getElement(attr(stats::terms(formula), "term.labels"), 1L)
)
```

**Arguments**

<code>data</code>	(data.frame/survey.design) a data frame or survey design object
<code>formula</code>	(formula) a formula
<code>method</code>	(string) string of function naming the function to be called, e.g. "glm". If function belongs to a library that is not attached, the package name must be specified in the package argument.
<code>method.args</code>	(named list) named list of arguments that will be passed to method. Note that this list may contain non-standard evaluation components. If you are wrapping this function in other functions, the argument must be passed in a way that does not evaluate the list, e.g. using rlang's embrace operator <code>{{ . }}</code> .
<code>package</code>	(string) string of package name that will be temporarily loaded when function specified in method is executed.
<code>response_type</code>	(string) string indicating whether the model outcome is 'continuous' or 'dichotomous'. When 'dichotomous', the call to <code>emmeans::emmeans()</code> is supplemented with argument <code>regrid="response"</code> .
<code>conf.level</code>	(scalar numeric) confidence level for confidence interval. Default is 0.95.
<code>primary_covariate</code>	(string) string indicating the primary covariate (typically the dichotomous treatment variable). Default is the first covariate listed in the formula.

**Value**

ARD data frame

**Examples**

```
ard_emmeans_mean_difference(
  data = mtcars,
  formula = mpg ~ am + cyl,
  method = "lm"
)

ard_emmeans_mean_difference(
  data = mtcars,
  formula = vs ~ am + mpg,
  method = "glm",
  method.args = list(family = binomial),
  response_type = "dichotomous"
)
```

---

```
ard_missing.survey.design
```

*ARD Missing Survey Statistics*

---

## Description

Compute Analysis Results Data (ARD) for statistics related to data missingness for survey objects

## Usage

```
## S3 method for class 'survey.design'
ard_missing(
  data,
  variables,
  by = NULL,
  statistic = everything() ~ c("N_obs", "N_miss", "N_nonmiss", "p_miss", "p_nonmiss",
    "N_obs_unweighted", "N_miss_unweighted", "N_nonmiss_unweighted", "p_miss_unweighted",
    "p_nonmiss_unweighted"),
  fmt_fn = NULL,
  stat_label = everything() ~ list(N_obs = "Total N", N_miss = "N Missing", N_nonmiss =
    "N not Missing", p_miss = "% Missing", p_nonmiss = "% not Missing",
    N_obs_unweighted = "Total N (unweighted)", N_miss_unweighted =
    "N Missing (unweighted)", N_nonmiss_unweighted = "N not Missing (unweighted)",
    p_miss_unweighted = "% Missing (unweighted)", p_nonmiss_unweighted =
    "% not Missing (unweighted)"),
  ...
)
```

## Arguments

data	(survey.design) a design object often created with <code>survey::svydesign()</code> .
variables	(tidy-select) columns to include in summaries.
by	(tidy-select) results are calculated for <b>all combinations</b> of the column specified and the variables. A single column may be specified.
statistic	(formula-list-selector) a named list, a list of formulas, or a single formula where the list element is a character vector of statistic names to include. See default value for options.
fmt_fn	(formula-list-selector) a named list, a list of formulas, or a single formula where the list element is a named list of functions (or the RHS of a formula), e.g. <code>list(mpg = list(mean = \(x) round(x, digits</code>
stat_label	(formula-list-selector) a named list, a list of formulas, or a single formula where the list element is either

a named list or a list of formulas defining the statistic labels, e.g. `everything() ~ list(mean = "Mean", sd = "SD")` or `everything() ~ list(mean ~ "Mean", sd ~ "SD")`.

... These dots are for future extensions and must be empty.

### Value

an ARD data frame of class 'card'

### Examples

```
svy_titanic <- survey::svydesign(~1, data = as.data.frame(Titanic), weights = ~Freq)
```

```
ard_missing(svy_titanic, variables = c(Class, Age), by = Survived)
```

---

ard_regression	<i>Regression ARD</i>
----------------	-----------------------

---

### Description

Function takes a regression model object and converts it to a ARD structure using the `broom.helpers` package.

### Usage

```
ard_regression(x, ...)
```

```
## Default S3 method:
```

```
ard_regression(x, tidy_fun = broom.helpers::tidy_with_broom_or_parameters, ...)
```

### Arguments

x	regression model object
...	Arguments passed to <code>broom.helpers::tidy_plus_plus()</code>
tidy_fun	(function) a tidier. Default is <code>broom.helpers::tidy_with_broom_or_parameters</code>

### Value

data frame

### Examples

```
lm(AGE ~ ARM, data = cards::ADSL) |>
  ard_regression(add_estimate_to_reference_rows = TRUE)
```

---

ard\_regression\_basic *Basic Regression ARD*


---

### Description

A function that takes a regression model and provides basic statistics in an ARD structure. The default output is simpler than `ard_regression()`. The function primarily matches regression terms to underlying variable names and levels. The default arguments used are

```
broom.helpers::tidy_plus_plus(
  add_reference_rows = FALSE,
  add_estimate_to_reference_rows = FALSE,
  add_n = FALSE,
  intercept = FALSE
)
```

### Usage

```
ard_regression_basic(
  x,
  tidy_fun = broom.helpers::tidy_with_broom_or_parameters,
  stats_to_remove = c("term", "var_type", "var_label", "var_class", "label",
    "contrasts_type", "contrasts", "var_nlevels"),
  ...
)
```

### Arguments

<code>x</code>	regression model object
<code>tidy_fun</code>	(function) a tidier. Default is <code>broom.helpers::tidy_with_broom_or_parameters</code>
<code>stats_to_remove</code>	(character) character vector of statistic names to remove. Default is <code>c("term", "var_type", "var_label", "var_class", "label", "contrasts_type", "contrasts", "var_nlevels")</code> .
<code>...</code>	Arguments passed to <code>broom.helpers::tidy_plus_plus()</code>

### Value

data frame

### Examples

```
lm(AGE ~ ARM, data = cards::ADSL) |>
  ard_regression_basic()
```

ard\_smd\_smd

*ARD Standardized Mean Difference***Description**

Standardized mean difference calculated via `smd::smd()` with `na.rm = TRUE`. Additionally, this function add a confidence interval to the SMD when `std.error=TRUE`, which the original `smd::smd()` does not include.

**Usage**

```
ard_smd_smd(data, by, variables, std.error = TRUE, conf.level = 0.95, ...)
```

**Arguments**

<code>data</code>	( <code>data.frame/survey.design</code> ) a data frame or object of class 'survey.design' (typically created with <code>survey::svydesign()</code> ).
<code>by</code>	( <code>tidy-select</code> ) column name to compare by.
<code>variables</code>	( <code>tidy-select</code> ) column names to be compared. Independent tests will be computed for each variable.
<code>std.error</code>	(scalar logical) Logical indicator for computing standard errors using <code>smd::compute_smd_var()</code> . Default is TRUE.
<code>conf.level</code>	(scalar numeric) confidence level for confidence interval. Default is 0.95.
<code>...</code>	arguments passed to <code>smd::smd()</code>

**Value**

ARD data frame

**Examples**

```
ard_smd_smd(cards::ADSL, by = SEX, variables = AGE)
ard_smd_smd(cards::ADSL, by = SEX, variables = AGEGR1)
```



---

ard_stats_anova	ARD ANOVA
-----------------	-----------

---

### Description

Prepare ANOVA results from the `stats::anova()` function. Users may pass a pre-calculated `stats::anova()` object or a list of formulas. In the latter case, the models will be constructed using the information passed and models will be passed to `stats::anova()`.

### Usage

```
ard_stats_anova(x, ...)

## S3 method for class 'anova'
ard_stats_anova(x, method_text = "ANOVA results from `stats::anova()`", ...)

## S3 method for class 'data.frame'
ard_stats_anova(
  x,
  formulas,
  method,
  method.args = list(),
  package = "base",
  method_text = "ANOVA results from `stats::anova()`",
  ...
)
```

### Arguments

<code>x</code>	( <code>anova</code> or <code>data.frame</code> ) an object of class 'anova' created with <code>stats::anova()</code> or a data frame
<code>...</code>	These dots are for future extensions and must be empty.
<code>method_text</code>	(string) string of the method used. Default is "ANOVA results from <code>stats::anova()</code> ". We provide the option to change this as <code>stats::anova()</code> can produce results from many types of models that may warrant a more precise description.
<code>formulas</code>	(list) a list of formulas
<code>method</code>	(string) string of function naming the function to be called, e.g. "glm". If function belongs to a library that is not attached, the package name must be specified in the package argument.
<code>method.args</code>	(named list) named list of arguments that will be passed to method.

Note that this list may contain non-standard evaluation components. If you are wrapping this function in other functions, the argument must be passed in a way that does not evaluate the list, e.g. using `rlang`'s embrace operator `{{ . }}`.

`package` (string)  
string of package name that will be temporarily loaded when function specified in method is executed.

### Details

When a list of formulas is supplied to `ard_stats_anova()`, these formulas along with information from other arguments, are used to construct models and pass those models to `stats::anova()`.

The models are constructed using `rlang::exec()`, which is similar to `do.call()`.

```
rlang::exec(.fn = method, formula = formula, data = data, !!!method.args)
```

The above function is executed in `withr::with_namespace(package)`, which allows for the use of `ard_stats_anova(method)` from packages, e.g. `package = 'lme4'` must be specified when `method = 'glmer'`. See example below.

### Value

ARD data frame

### Examples

```
anova(
  lm(mpg ~ am, mtcars),
  lm(mpg ~ am + hp, mtcars)
) |>
ard_stats_anova()
```

```
ard_stats_anova(
  x = mtcars,
  formulas = list(am ~ mpg, am ~ mpg + hp),
  method = "glm",
  method.args = list(family = binomial)
)
```

```
ard_stats_anova(
  x = mtcars,
  formulas = list(am ~ 1 + (1 | vs), am ~ mpg + (1 | vs)),
  method = "glmer",
  method.args = list(family = binomial),
  package = "lme4"
)
```

---

ard_stats_aov	<i>ARD ANOVA</i>
---------------	------------------

---

**Description**

Analysis results data for Analysis of Variance. Calculated with `stats::aov()`

**Usage**

```
ard_stats_aov(formula, data, ...)
```

**Arguments**

formula	A formula specifying the model.
data	A data frame in which the variables specified in the formula will be found. If missing, the variables are searched for in the standard way.
...	arguments passed to <code>stats::aov(...)</code>

**Value**

ARD data frame

**Examples**

```
ard_stats_aov(AGE ~ ARM, data = cards::ADSL)
```

---

ard_stats_chisq_test	<i>ARD Chi-squared Test</i>
----------------------	-----------------------------

---

**Description**

Analysis results data for Pearson's Chi-squared Test. Calculated with `chisq.test(x = data[[variable]], y = data[[by]], ...)`

**Usage**

```
ard_stats_chisq_test(data, by, variables, ...)
```

**Arguments**

data	(data.frame) a data frame.
by	(tidy-select) column name to compare by.
variables	(tidy-select) column names to be compared. Independent tests will be computed for each variable.
...	additional arguments passed to <code>chisq.test(...)</code>

**Value**

ARD data frame

**Examples**

```
cards::ADSL |>
  ard_stats_chisq_test(by = "ARM", variables = "AGEGR1")
```

---

ard\_stats\_fisher\_test *ARD Fisher's Exact Test*

---

**Description**

Analysis results data for Fisher's Exact Test. Calculated with `fisher.test(x = data[[variable]], y = data[[by]], ...)`

**Usage**

```
ard_stats_fisher_test(data, by, variables, conf.level = 0.95, ...)
```

**Arguments**

data	(data.frame) a data frame.
by	(tidy-select) column name to compare by
variables	(tidy-select) column names to be compared. Independent tests will be computed for each variable.
conf.level	(scalar numeric) confidence level for confidence interval. Default is 0.95.
...	additional arguments passed to <code>fisher.test(...)</code>

**Value**

ARD data frame

**Examples**

```
cards::ADSL[1:30, ] |>
  ard_stats_fisher_test(by = "ARM", variables = "AGEGR1")
```

---

ard\_stats\_kruskal\_test

*ARD Kruskal-Wallis Test*

---

**Description**

Analysis results data for Kruskal-Wallis Rank Sum Test.

Calculated with `kruskal.test(data[[variable]], data[[by]], ...)`

**Usage**

```
ard_stats_kruskal_test(data, by, variables)
```

**Arguments**

data	(data.frame) a data frame.
by	(tidy-select) column name to compare by.
variables	(tidy-select) column names to be compared. Independent tests will be computed for each variable.

**Value**

ARD data frame

**Examples**

```
cards::ADSL |>
  ard_stats_kruskal_test(by = "ARM", variables = "AGE")
```

---

`ard_stats_mcnemar_test`*ARD McNemar's Test*

---

## Description

Analysis results data for McNemar's statistical test. We have two functions depending on the structure of the data.

- `ard_stats_mcnemar_test()` is the structure expected by `stats::mcnemar.test()`
- `ard_stats_mcnemar_test_long()` is one row per ID per group

## Usage

```
ard_stats_mcnemar_test(data, by, variables, ...)
```

```
ard_stats_mcnemar_test_long(data, by, variables, id, ...)
```

## Arguments

<code>data</code>	( <code>data.frame</code> ) a data frame. See below for details.
<code>by</code>	( <code>tidy-select</code> ) column name to compare by.
<code>variables</code>	( <code>tidy-select</code> ) column names to be compared. Independent tests will be computed for each variable.
<code>...</code>	arguments passed to <code>stats::mcnemar.test(...)</code>
<code>id</code>	( <code>tidy-select</code> ) column name of the subject or participant ID

## Details

For the `ard_stats_mcnemar_test()` function, the data is expected to be one row per subject. The data is passed as `stats::mcnemar.test(x = data[[variable]], y = data[[by]], ...)`. Please use `table(x = data[[variable]], y = data[[by]])` to check the contingency table.

## Value

ARD data frame

**Examples**

```
cards::ADSL |>
  ard_stats_mcnemar_test(by = "SEX", variables = "EFFFL")

set.seed(1234)
cards::ADSL[c("USUBJID", "TRT01P")] |>
  dplyr::mutate(TYPE = "PLANNED") |>
  dplyr::rename(TRT01 = TRT01P) %>%
  dplyr::bind_rows(dplyr::mutate(., TYPE = "ACTUAL", TRT01 = sample(TRT01))) |>
  ard_stats_mcnemar_test_long(
    by = TYPE,
    variable = TRT01,
    id = USUBJID
  )
```

---

ard\_stats\_mood\_test    *ARD Mood Test*

---

**Description**

Analysis results data for Mood two sample test of scale. Note this not to be confused with the Brown-Mood test of medians.

**Usage**

```
ard_stats_mood_test(data, by, variables, ...)
```

**Arguments**

data	(data.frame) a data frame. See below for details.
by	(tidy-select) column name to compare by.
variables	(tidy-select) column name to be compared. Independent tests will be run for each variable.
...	arguments passed to mood.test(...)

**Details**

For the `ard_stats_mood_test()` function, the data is expected to be one row per subject. The data is passed as `mood.test(data[[variable]] ~ data[[by]], ...)`.

**Value**

ARD data frame

**Examples**

```
cards::ADSL |>
  ard_stats_mood_test(by = "SEX", variables = "AGE")
```

---

ard\_stats\_oneway\_test *ARD One-way Test*

---

**Description**

Analysis results data for Testing Equal Means in a One-Way Layout. calculated with `oneway.test()`

**Usage**

```
ard_stats_oneway_test(formula, data, ...)
```

**Arguments**

formula	a formula of the form <code>lhs ~ rhs</code> where <code>lhs</code> gives the sample values and <code>rhs</code> the corresponding groups.
data	an optional matrix or data frame (or similar: see <a href="#">model.frame</a> ) containing the variables in the formula <code>formula</code> . By default the variables are taken from <code>environment(formula)</code> .
...	additional arguments passed to <code>oneway.test(...)</code>

**Value**

ARD data frame

**Examples**

```
ard_stats_oneway_test(AGE ~ ARM, data = cards::ADSL)
```

---

ard\_stats\_poisson\_test  
*ARD Poisson Test*

---

**Description**

Analysis results data for exact tests of a simple null hypothesis about the rate parameter in Poisson distribution, or the comparison of two rate parameters.



**Usage**

```
ard_stats_poisson_test(
  data,
  variables,
  na.rm = TRUE,
  by = NULL,
  conf.level = 0.95,
  ...
)
```

**Arguments**

data	(data.frame) a data frame. See below for details.
variables	(tidy-select) names of the event and time variables (in that order) to be used in computations. Must be of length 2.
na.rm	(scalar logical) whether missing values should be removed before computations. Default is TRUE.
by	(tidy-select) optional column name to compare by.
conf.level	(scalar numeric) confidence level for confidence interval. Default is 0.95.
...	arguments passed to <code>poisson.test()</code> .

**Details**

- For the `ard_stats_poisson_test()` function, the data is expected to be one row per subject.
- If `by` is not specified, an exact Poisson test of the rate parameter will be performed. Otherwise, a Poisson comparison of two rate parameters will be performed on the levels of `by`. If `by` has more than 2 levels, an error will occur.

**Value**

an ARD data frame of class 'card'

**Examples**

```
# Exact test of rate parameter against null hypothesis
cards::ADTTE |>
  ard_stats_poisson_test(variables = c(CNSR, AVAL))

# Comparison test of ratio of 2 rate parameters against null hypothesis
cards::ADTTE |>
  dplyr::filter(TRTA %in% c("Placebo", "Xanomeline High Dose")) |>
  ard_stats_poisson_test(by = TRTA, variables = c(CNSR, AVAL))
```

---

ard\_stats\_prop\_test     *ARD 2-sample proportion test*

---

### Description

Analysis results data for a 2-sample test or proportions using `stats::prop.test()`.

### Usage

```
ard_stats_prop_test(data, by, variables, conf.level = 0.95, ...)
```

### Arguments

data	(data.frame) a data frame.
by	(tidy-select) column name to compare by
variables	(tidy-select) column names to be compared. Must be a binary column coded as TRUE/FALSE or 1/0. Independent tests will be computed for each variable.
conf.level	(scalar numeric) confidence level for confidence interval. Default is 0.95.
...	arguments passed to <code>prop.test(...)</code>

### Value

ARD data frame

### Examples

```
mtcars |>
  ard_stats_prop_test(by = vs, variables = am)
```

---

ard\_stats\_t\_test     *ARD t-test*

---

### Description

Analysis results data for paired and non-paired t-tests.

### Usage

```
ard_stats_t_test(data, variables, by = NULL, conf.level = 0.95, ...)
```

```
ard_stats_paired_t_test(data, by, variables, id, conf.level = 0.95, ...)
```

**Arguments**

data	(data.frame) a data frame. See below for details.
variables	(tidy-select) column names to be compared. Independent t-tests will be computed for each variable.
by	(tidy-select) optional column name to compare by.
conf.level	(scalar numeric) confidence level for confidence interval. Default is 0.95.
...	arguments passed to t.test()
id	(tidy-select) column name of the subject or participant ID

**Details**

For the `ard_stats_t_test()` function, the data is expected to be one row per subject. The data is passed as `t.test(data[[variable]] ~ data[[by]], paired = FALSE, ...)`.

For the `ard_stats_paired_t_test()` function, the data is expected to be one row per subject per by level. Before the t-test is calculated, the data are reshaped to a wide format to be one row per subject. The data are then passed as `t.test(x = data_wide[[<by level 1>]], y = data_wide[[<by level 2>]], paired =`

**Value**

ARD data frame

**Examples**

```
cards::ADSL |>
  dplyr::filter(ARM %in% c("Placebo", "Xanomeline High Dose")) |>
  ard_stats_t_test(by = ARM, variables = c(AGE, BMIBL))

# constructing a paired data set,
# where patients receive both treatments
cards::ADSL[c("ARM", "AGE")] |>
  dplyr::filter(ARM %in% c("Placebo", "Xanomeline High Dose")) |>
  dplyr::mutate(.by = ARM, USUBJID = dplyr::row_number()) |>
  dplyr::arrange(USUBJID, ARM) |>
  ard_stats_paired_t_test(by = ARM, variables = AGE, id = USUBJID)
```

---

```
ard_stats_t_test_onesample
```

*ARD one-sample t-test*

---

## Description

Analysis results data for one-sample t-tests. Result may be stratified by including the `by` argument.

## Usage

```
ard_stats_t_test_onesample(  
  data,  
  variables,  
  by = dplyr::group_vars(data),  
  conf.level = 0.95,  
  ...  
)
```

## Arguments

<code>data</code>	( <code>data.frame</code> ) a data frame. See below for details.
<code>variables</code>	( <a href="#">tidy-select</a> ) column names to be analyzed. Independent t-tests will be computed for each variable.
<code>by</code>	( <a href="#">tidy-select</a> ) optional column name to stratify results by.
<code>conf.level</code>	(scalar numeric) confidence level for confidence interval. Default is 0.95.
<code>...</code>	arguments passed to <code>t.test()</code>

## Value

ARD data frame

## Examples

```
cards::ADSL |>  
  ard_stats_t_test_onesample(by = ARM, variables = AGE)
```

---

 ard\_stats\_wilcox\_test *ARD Wilcoxon Rank-Sum Test*


---

**Description**

Analysis results data for paired and non-paired Wilcoxon Rank-Sum tests.

**Usage**

```
ard_stats_wilcox_test(data, variables, by = NULL, conf.level = 0.95, ...)
ard_stats_paired_wilcox_test(data, by, variables, id, conf.level = 0.95, ...)
```

**Arguments**

data	(data.frame) a data frame. See below for details.
variables	(tidy-select) column names to be compared. Independent tests will be computed for each variable.
by	(tidy-select) optional column name to compare by.
conf.level	(scalar numeric) confidence level for confidence interval. Default is 0.95.
...	arguments passed to wilcox.test(...)
id	(tidy-select) column name of the subject or participant ID.

**Details**

For the `ard_stats_wilcox_test()` function, the data is expected to be one row per subject. The data is passed as `wilcox.test(data[[variable]] ~ data[[by]], paired = FALSE, ...)`.

For the `ard_stats_paired_wilcox_test()` function, the data is expected to be one row per subject per by level. Before the test is calculated, the data are reshaped to a wide format to be one row per subject. The data are then passed as `wilcox.test(x = data_wide[<by level 1>], y = data_wide[<by level 2>])`.

**Value**

ARD data frame

**Examples**

```
cards::ADSL |>
  dplyr::filter(ARM %in% c("Placebo", "Xanomeline High Dose")) |>
  ard_stats_wilcox_test(by = "ARM", variables = "AGE")
```

```
# constructing a paired data set,
# where patients receive both treatments
cards::ADSL[c("ARM", "AGE")] |>
  dplyr::filter(ARM %in% c("Placebo", "Xanomeline High Dose")) |>
  dplyr::mutate(.by = ARM, USUBJID = dplyr::row_number()) |>
  dplyr::arrange(USUBJID, ARM) |>
  ard_stats_paired_wilcox_test(by = ARM, variables = AGE, id = USUBJID)
```

---

```
ard_stats_wilcox_test_onesample
```

*ARD one-sample Wilcox Rank-sum*

---

## Description

Analysis results data for one-sample Wilcox Rank-sum. Result may be stratified by including the `by` argument.

## Usage

```
ard_stats_wilcox_test_onesample(
  data,
  variables,
  by = dplyr::group_vars(data),
  conf.level = 0.95,
  ...
)
```

## Arguments

<code>data</code>	( <code>data.frame</code> ) a data frame. See below for details.
<code>variables</code>	( <a href="#">tidy-select</a> ) column names to be analyzed. Independent Wilcox Rank-sum tests will be computed for each variable.
<code>by</code>	( <a href="#">tidy-select</a> ) optional column name to stratify results by.
<code>conf.level</code>	( <code>scalar numeric</code> ) confidence level for confidence interval. Default is 0.95.
<code>...</code>	arguments passed to <code>wilcox.test(...)</code>

## Value

ARD data frame

**Examples**

```
cards::ADSL |>
  ard_stats_wilcox_test_onesample(by = ARM, variables = AGE)
```

---

ard\_survey\_svychisq     *ARD Survey Chi-Square Test*

---

**Description**

Analysis results data for survey Chi-Square test using `survey::svychisq()`. Only two-way comparisons are supported.

**Usage**

```
ard_survey_svychisq(data, by, variables, statistic = "F", ...)
```

**Arguments**

<code>data</code>	( <code>survey.design</code> ) a survey design object often created with the <code>{survey}</code> package
<code>by</code>	( <code>tidy-select</code> ) column name to compare by.
<code>variables</code>	( <code>tidy-select</code> ) column names to be compared. Independent tests will be computed for each variable.
<code>statistic</code>	(character) statistic used to estimate Chisq p-value. Default is the Rao-Scott second-order correction ("F"). See <code>survey::svychisq</code> for available statistics options.
<code>...</code>	arguments passed to <code>survey::svychisq()</code> .

**Value**

ARD data frame

**Examples**

```
data(api, package = "survey")
dclus1 <- survey::svydesign(id = ~dnum, weights = ~pw, data = apiclus1, fpc = ~fpc)

ard_survey_svychisq(dclus1, variables = sch.wide, by = comp.imp, statistic = "F")
```

---

`ard_survey_svyranktest`*ARD Survey rank test*

---

## Description

Analysis results data for survey wilcox test using `survey::svyranktest()`.

## Usage

```
ard_survey_svyranktest(data, by, variables, test, ...)
```

## Arguments

<code>data</code>	( <code>survey.design</code> ) a survey design object often created with <code>survey::svydesign()</code>
<code>by</code>	( <code>tidy-select</code> ) column name to compare by
<code>variables</code>	( <code>tidy-select</code> ) column names to be compared. Independent tests will be run for each variable.
<code>test</code>	( <code>string</code> ) a string to denote which rank test to use: "wilcoxon", "vanderWaerden", "median", "KruskalWallis"
<code>...</code>	arguments passed to <code>survey::svyranktest()</code>

## Value

ARD data frame

## Examples

```
data(api, package = "survey")
dclus2 <- survey::svydesign(id = ~ dnum + snum, fpc = ~ fpc1 + fpc2, data = apiclus2)

ard_survey_svyranktest(dclus2, variables = enroll, by = comp.imp, test = "wilcoxon")
ard_survey_svyranktest(dclus2, variables = enroll, by = comp.imp, test = "vanderWaerden")
ard_survey_svyranktest(dclus2, variables = enroll, by = comp.imp, test = "median")
ard_survey_svyranktest(dclus2, variables = enroll, by = comp.imp, test = "KruskalWallis")
```



---

ard\_survey\_svytttest *ARD Survey t-test*


---

**Description**

Analysis results data for survey t-test using `survey::svytttest()`.

**Usage**

```
ard_survey_svytttest(data, by, variables, conf.level = 0.95, ...)
```

**Arguments**

<code>data</code>	( <code>survey.design</code> ) a survey design object often created with <code>survey::svydesign()</code>
<code>by</code>	( <code>tidy-select</code> ) column name to compare by
<code>variables</code>	( <code>tidy-select</code> ) column names to be compared. Independent tests will be run for each variable.
<code>conf.level</code>	( <code>double</code> ) confidence level of the returned confidence interval. Must be between $c(0, 1)$ . Default is 0.95
<code>...</code>	arguments passed to <code>survey::svytttest()</code>

**Value**

ARD data frame

**Examples**

```
data(api, package = "survey")
dclus2 <- survey::svydesign(id = ~ dnum + snum, fpc = ~ fpc1 + fpc2, data = apiclus2)

ard_survey_svytttest(dclus2, variables = enroll, by = comp.imp, conf.level = 0.9)
```

---

ard\_survival\_survdiff *ARD for Difference in Survival*


---

**Description**

Analysis results data for comparison of survival using `survival::survdiff()`.

**Usage**

```
ard_survival_survdiff(formula, data, rho = 0, ...)
```

**Arguments**

formula	(formula) a formula
data	(data.frame) a data frame
rho	(scalar numeric) numeric scalar passed to <code>survival::survdiff(rho)</code> . Default is <code>rho=0</code> .
...	additional arguments passed to <code>survival::survdiff()</code>

**Value**

an ARD data frame of class 'card'

**Examples**

```
library(survival)
library(ggsurvfit)

ard_survival_survdiff(Surv_CNSR(AVAL, CNSR) ~ TRTA, data = cards::ADTTE)
```

---

ard\_survival\_survfit *ARD Survival Estimates*

---

**Description**

Analysis results data for survival quantiles and x-year survival estimates, extracted from a `survival::survfit()` model.

**Usage**

```
ard_survival_survfit(x, ...)

## S3 method for class 'survfit'
ard_survival_survfit(x, times = NULL, probs = NULL, type = NULL, ...)

## S3 method for class 'data.frame'
ard_survival_survfit(
  x,
  y,
  variables,
  times = NULL,
  probs = NULL,
  type = NULL,
  method.args = list(conf.int = 0.95),
  ...
)
```

**Arguments**

<code>x</code>	( <code>survfit</code> or <code>data.frame</code> ) an object of class <code>survfit</code> created with <code>survival::survfit()</code> or a data frame. See below for details.								
<code>...</code>	These dots are for future extensions and must be empty.								
<code>times</code>	(numeric) a vector of times for which to return survival probabilities.								
<code>probs</code>	(numeric) a vector of probabilities with values in (0,1) specifying the survival quantiles to return.								
<code>type</code>	(string or <code>NULL</code> ) type of statistic to report. Available for Kaplan-Meier time estimates only, otherwise type is ignored. Default is <code>NULL</code> . Must be one of the following: <table data-bbox="646 768 976 894"> <thead> <tr> <th>type</th> <th>transformation</th> </tr> </thead> <tbody> <tr> <td>"survival"</td> <td><math>x</math></td> </tr> <tr> <td>"risk"</td> <td><math>1 - x</math></td> </tr> <tr> <td>"cumhaz"</td> <td><math>-\log(x)</math></td> </tr> </tbody> </table>	type	transformation	"survival"	$x$	"risk"	$1 - x$	"cumhaz"	$-\log(x)$
type	transformation								
"survival"	$x$								
"risk"	$1 - x$								
"cumhaz"	$-\log(x)$								
<code>y</code>	( <code>Surv</code> or string) an object of class <code>Surv</code> created using <code>survival::Surv()</code> . This object will be passed as the left-hand side of the formula constructed and passed to <code>survival::survfit()</code> . This object can also be passed as a string.								
<code>variables</code>	(character) stratification variables to be passed as the right-hand side of the formula constructed and passed to <code>survival::survfit()</code> .								
<code>method.args</code>	(named list) named list of arguments that will be passed to <code>survival::survfit()</code> .								

**Details**

- Only one of either the `times` or `probs` parameters can be specified.
- Times should be provided using the same scale as the time variable used to fit the provided survival fit model.

**Value**

an ARD data frame of class 'card'

**Formula Specification**

When passing a `survival::survfit()` object to `ard_survival_survfit()`, the `survfit()` call must use an evaluated formula and not a stored formula. Including a proper formula in the call allows the function to accurately identify all variables included in the estimation. See below for examples:

```

library(cardx)
library(survival)

# include formula in `survfit()` call
survfit(Surv(time, status) ~ sex, lung) |> ard_survival_survfit(time = 500)

# you can also pass a data frame to `ard_survival_survfit()` as well.
lung |>
  ard_survival_survfit(y = Surv(time, status), variables = "sex", time = 500)

```

You **cannot**, however, pass a stored formula, e.g. `survfit(my_formula, lung)`, but you can use stored formulas with `rlang::inject(survfit(!my_formula, lung))`.

### Variable Classes

When the `survfit` method is called, the class of the stratifying variables will be returned as a factor.

When the data frame method is called, the original classes are retained in the resulting ARD.

### Examples

```

library(survival)
library(ggsurvfit)

survfit(Surv_CNSR(AVAL, CNSR) ~ TRTA, data = cards::ADTTE) |>
  ard_survival_survfit(times = c(60, 180))

survfit(Surv_CNSR(AVAL, CNSR) ~ TRTA, data = cards::ADTTE, conf.int = 0.90) |>
  ard_survival_survfit(probs = c(0.25, 0.5, 0.75))

cards::ADTTE |>
  ard_survival_survfit(y = Surv_CNSR(AVAL, CNSR), variables = c("TRTA", "SEX"), times = 90)

# Competing Risks Example -----
set.seed(1)
ADTTE_MS <- cards::ADTTE %>%
  dplyr::mutate(
    CNSR = dplyr::case_when(
      CNSR == 0 ~ "censor",
      runif(dplyr::n()) < 0.5 ~ "death from cancer",
      TRUE ~ "death other causes"
    ) %>% factor()
  )

survfit(Surv(AVAL, CNSR) ~ TRTA, data = ADTTE_MS) %>%
  ard_survival_survfit(times = c(60, 180))

```

---

`ard_survival_survfit_diff`*ARD Survival Differences*

---

**Description**

Calculate differences in the Kaplan-Meier estimator of survival using the results from `survival::survfit()`.

**Usage**

```
ard_survival_survfit_diff(x, times, conf.level = 0.95)
```

**Arguments**

<code>x</code>	(survfit) object of class 'survfit' typically created with <code>survival::survfit()</code>
<code>times</code>	(numeric) a vector of times for which to return survival probabilities.
<code>conf.level</code>	(scalar numeric) confidence level for confidence interval. Default is 0.95.

**Value**

an ARD data frame of class 'card'

**Examples**

```
library(ggsurvfit)
library(survival)

survfit(Surv_CNSR() ~ TRTA, data = cards::ADTTE) |>
  ard_survival_survfit_diff(times = c(25, 50))
```

---

`ard_total_n.survey.design`*ARD Total N*

---

**Description**

Returns the total N for a survey object. The placeholder variable name returned in the object is `"..ard_total_n.."`

**Usage**

```
## S3 method for class 'survey.design'
ard_total_n(data, ...)
```

**Arguments**

data (survey.design)  
a design object often created with `survey::svydesign()`.

... These dots are for future extensions and must be empty.

**Value**

an ARD data frame of class 'card'

**Examples**

```
svy_titanic <- survey::svydesign(~1, data = as.data.frame(Titanic), weights = ~Freq)
ard_total_n(svy_titanic)
```

---

construction\_helpers *Construction Helpers*

---

**Description**

These functions help construct calls to various types of models.

**Usage**

```
construct_model(data, ...)
```

```
## S3 method for class 'data.frame'
construct_model(
  data,
  formula,
  method,
  method.args = list(),
  package = "base",
  env = caller_env(),
  ...
)
```

```
## S3 method for class 'survey.design'
construct_model(
  data,
  formula,
  method,
  method.args = list(),
  package = "survey",
  env = caller_env(),
  ...
)
```

```

)

reformulate2(
  termlabels,
  response = NULL,
  intercept = TRUE,
  env = parent.frame(),
  pattern_term = NULL,
  pattern_response = NULL
)

bt(x, pattern = NULL)

bt_strip(x)

```

### Arguments

data	<ul style="list-style-type: none"> <li>• <code>construct_model.data.frame()</code> (<code>data.frame</code>) a data frame</li> <li>• <code>construct_model.survey.design()</code> (<code>survey.design</code>) a survey design object</li> </ul>
...	These dots are for future extensions and must be empty.
formula	(formula) a formula
method	(string) string of function naming the function to be called, e.g. "glm". If function belongs to a library that is not attached, the package name must be specified in the package argument.
method.args	(named list) named list of arguments that will be passed to method. Note that this list may contain non-standard evaluation components. If you are wrapping this function in other functions, the argument must be passed in a way that does not evaluate the list, e.g. using <code>rlang</code> 's embrace operator <code>{{ . }}</code> .
package	(string) string of package name that will be temporarily loaded when function specified in method is executed.
env	The environment in which to evaluate <code>expr</code> . This environment is not applicable for quosures because they have their own environments.
termlabels	character vector giving the right-hand side of a model formula. Cannot be zero-length.
response	character string, symbol or call giving the left-hand side of a model formula, or <code>NULL</code> .
intercept	logical: should the formula have an intercept?
x	(character) character vector, typically of variable names
pattern, pattern_term, pattern_response	DEPRECATED

**Details**

- `construct_model()`: Builds models of the form `method(data = data, formula = formula, method.args!!!)`. If the package argument is specified, that package is temporarily attached when the model is evaluated.
- `reformulate2()`: This is a copy of `reformulate()` except that variable names that contain a space are wrapped in backticks.
- `bt()`: Adds backticks to a character vector.
- `bt_strip()`: Removes backticks from a string if it begins and ends with a backtick.

**Value**

depends on the calling function

**Examples**

```
construct_model(
  data = mtcars,
  formula = am ~ mpg + (1 | vs),
  method = "glmer",
  method.args = list(family = binomial),
  package = "lme4"
) |>
broom.mixed::tidy()

construct_model(
  data = mtcars |> dplyr::rename(`M P G` = mpg),
  formula = reformulate2(c("M P G", "cyl"), response = "hp"),
  method = "lm"
) |>
ard_regression() |>
dplyr::filter(stat_name %in% c("term", "estimate", "p.value"))
```

---

proportion\_ci

*Functions for Calculating Proportion Confidence Intervals*


---

**Description**

Functions to calculate different proportion confidence intervals for use in `ard_proportion()`.

**Usage**

```
proportion_ci_wald(x, conf.level = 0.95, correct = FALSE)
```

```
proportion_ci_wilson(x, conf.level = 0.95, correct = FALSE)
```

```
proportion_ci_clopper_pearson(x, conf.level = 0.95)
```



```

proportion_ci_agresti_coull(x, conf.level = 0.95)

proportion_ci_jeffreys(x, conf.level = 0.95)

proportion_ci_strat_wilson(
  x,
  strata,
  weights = NULL,
  conf.level = 0.95,
  max.iterations = 10L,
  correct = FALSE
)

is_binary(x)

```

### Arguments

x	vector of a binary values, i.e. a logical vector, or numeric with values c(0, 1)
conf.level	(numeric) a scalar in (0, 1) indicating the confidence level. Default is 0.95
correct	(flag) include the continuity correction. For further information, see for example <a href="#">stats::prop.test()</a> .
strata	(factor) variable with one level per stratum and same length as x.
weights	(numeric or NULL) weights for each level of the strata. If NULL, they are estimated using the iterative algorithm that minimizes the weighted squared length of the confidence interval.
max.iterations	(count) maximum number of iterations for the iterative procedure used to find estimates of optimal weights.

### Value

Confidence interval of a proportion.

### Functions

- `proportion_ci_wald()`: Calculates the Wald interval by following the usual textbook definition for a single proportion confidence interval using the normal approximation.

$$\hat{p} \pm z_{\alpha/2} \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}$$

- `proportion_ci_wilson()`: Calculates the Wilson interval by calling [stats::prop.test\(\)](#). Also referred to as Wilson score interval.

$$\frac{\hat{p} + \frac{z_{\alpha/2}^2}{2n} \pm z_{\alpha/2} \sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z_{\alpha/2}^2}{4n^2}}}{1 + \frac{z_{\alpha/2}^2}{n}}$$

- `proportion_ci_clopper_pearson()`: Calculates the Clopper-Pearson interval by calling `stats::binom.test()`. Also referred to as the exact method.

$$\left( \frac{k}{n} \pm z_{\alpha/2} \sqrt{\frac{\frac{k}{n}(1-\frac{k}{n})}{n} + \frac{z_{\alpha/2}^2}{4n^2}} \right) / \left( 1 + \frac{z_{\alpha/2}^2}{n} \right)$$

- `proportion_ci_agresti_coull()`: Calculates the Agresti-Coull interval (created by Alan Agresti and Brent Coull) by (for 95% CI) adding two successes and two failures to the data and then using the Wald formula to construct a CI.

$$\left( \frac{\tilde{p} + z_{\alpha/2}^2/2}{n + z_{\alpha/2}^2} \pm z_{\alpha/2} \sqrt{\frac{\tilde{p}(1-\tilde{p})}{n} + \frac{z_{\alpha/2}^2}{4n^2}} \right)$$

- `proportion_ci_jeffreys()`: Calculates the Jeffreys interval, an equal-tailed interval based on the non-informative Jeffreys prior for a binomial proportion.

$$\left( \text{Beta} \left( \frac{k}{2} + \frac{1}{2}, \frac{n-k}{2} + \frac{1}{2} \right)_{\alpha}, \text{Beta} \left( \frac{k}{2} + \frac{1}{2}, \frac{n-k}{2} + \frac{1}{2} \right)_{1-\alpha} \right)$$

- `proportion_ci_strat_wilson()`: Calculates the stratified Wilson confidence interval for unequal proportions as described in Xin YA, Su XG. Stratified Wilson and Newcombe confidence intervals for multiple binomial proportions. *Statistics in Biopharmaceutical Research*. 2010;2(3).

$$\frac{\hat{p}_j + \frac{z_{\alpha/2}^2}{2n_j} \pm z_{\alpha/2} \sqrt{\frac{\hat{p}_j(1-\hat{p}_j)}{n_j} + \frac{z_{\alpha/2}^2}{4n_j^2}}}{1 + \frac{z_{\alpha/2}^2}{n_j}}$$

- `is_binary()`: Helper to determine if vector is binary (logical or 0/1)

## Examples

```
x <- c(
  TRUE, TRUE, TRUE, TRUE, TRUE,
  FALSE, FALSE, FALSE, FALSE, FALSE
)
```

```
proportion_ci_wald(x, conf.level = 0.9)
proportion_ci_wilson(x, correct = TRUE)
proportion_ci_clopper_pearson(x)
proportion_ci_agresti_coull(x)
proportion_ci_jeffreys(x)
```

```
# Stratified Wilson confidence interval with unequal probabilities
```

```
set.seed(1)
rsp <- sample(c(TRUE, FALSE), 100, TRUE)
strata_data <- data.frame(
  "f1" = sample(c("a", "b"), 100, TRUE),
  "f2" = sample(c("x", "y", "z"), 100, TRUE),
  stringsAsFactors = TRUE
)
strata <- interaction(strata_data)
n_strata <- ncol(table(rsp, strata)) # Number of strata

proportion_ci_strat_wilson(
  x = rsp, strata = strata,
  conf.level = 0.90
)

# Not automatic setting of weights
proportion_ci_strat_wilson(
  x = rsp, strata = strata,
  weights = rep(1 / n_strata, n_strata),
  conf.level = 0.90
)
```

# Index

`aod::wald.test()`, 3  
`ard_aod_wald_test`, 3  
`ard_attributes.survey.design`, 4  
`ard_car_anova`, 5  
`ard_car_vif`, 5  
`ard_categorical.survey.design`, 6  
`ard_categorical_ci`, 7  
`ard_categorical_ci.survey.design`, 9  
`ard_categorical_max`, 10  
`ard_continuous.survey.design`, 11  
`ard_continuous_ci`, 13  
`ard_continuous_ci.survey.design`, 14  
`ard_dichotomous.survey.design`, 15  
`ard_effectsize_cohens_d`, 16  
`ard_effectsize_hedges_g`, 18  
`ard_effectsize_paired_cohens_d`  
  (`ard_effectsize_cohens_d`), 16  
`ard_effectsize_paired_hedges_g`  
  (`ard_effectsize_hedges_g`), 18  
`ard_emmeans_mean_difference`, 19  
`ard_missing.survey.design`, 21  
`ard_regression`, 22  
`ard_regression()`, 23  
`ard_regression_basic`, 23  
`ard_smd_smd`, 24  
`ard_stats_anova`, 25  
`ard_stats_aov`, 27  
`ard_stats_chisq_test`, 27  
`ard_stats_fisher_test`, 28  
`ard_stats_kruskal_test`, 29  
`ard_stats_mcnemar_test`, 30  
`ard_stats_mcnemar_test_long`  
  (`ard_stats_mcnemar_test`), 30  
`ard_stats_mood_test`, 31  
`ard_stats_oneway_test`, 32  
`ard_stats_paired_t_test`  
  (`ard_stats_t_test`), 34  
`ard_stats_paired_wilcox_test`  
  (`ard_stats_wilcox_test`), 37  
`ard_stats_poisson_test`, 32  
`ard_stats_prop_test`, 34  
`ard_stats_t_test`, 34  
`ard_stats_t_test_onesample`, 36  
`ard_stats_wilcox_test`, 37  
`ard_stats_wilcox_test_onesample`, 38  
`ard_survey_svychisq`, 39  
`ard_survey_svyranktest`, 40  
`ard_survey_svyttest`, 41  
`ard_survival_survdiff`, 41  
`ard_survival_survfit`, 42  
`ard_survival_survfit_diff`, 45  
`ard_total_n.survey.design`, 45  
  
`broom.helpers::tidy_plus_plus()`, 22, 23  
`broom.helpers::tidy_with_broom_or_parameters`,  
  3, 22, 23  
`bt` (`construction_helpers`), 46  
`bt_strip` (`construction_helpers`), 46  
  
`car::Anova()`, 5  
`car::vif()`, 5  
`cards::ard_categorical()`, 11  
`construct_model` (`construction_helpers`),  
  46  
`construction_helpers`, 46  
  
`effectsize::cohens_d()`, 16  
`effectsize::hedges_g()`, 18  
  
`is_binary` (`proportion_ci`), 48  
  
`model.frame`, 32  
  
`poisson.test()`, 33  
`proportion_ci`, 48  
`proportion_ci_agresti_coull`  
  (`proportion_ci`), 48  
`proportion_ci_clopper_pearson`  
  (`proportion_ci`), 48

proportion\_ci\_jeffreys (proportion\_ci),  
48

proportion\_ci\_strat\_wilson  
(proportion\_ci), 48

proportion\_ci\_wald (proportion\_ci), 48

proportion\_ci\_wilson (proportion\_ci), 48

reformulate2 (construction\_helpers), 46

smd::smd(), 24

stats::binom.test(), 50

stats::mcnemar.test(), 30

stats::prop.test(), 34, 49

survey::svychisq, 39

survey::svychisq(), 39

survey::svyciprop(), 9

survey::svydesign(), 4, 6, 9, 12, 14, 16, 21,  
24, 40, 41, 46

survey::svyranktest(), 40

survey::svyttest(), 41

survival::Surv(), 43

survival::survdiff(), 41

survival::survfit(), 42, 43, 45