

Package: baseverse (via r-universe)

May 31, 2026

Title Modern Base-R Functions

Version 0.1.4

Description Includes modern base-R functions. Functions beginning with `p_` are wrapper functions for existing base-R functions, supporting native piping. Other functions are wrapper functions for core base-R features, including bracket notation and dollar-sign notation. `base_match()` and `base_when()` mimic `case_match()` and `case_when()` from 'dplyr' but return a factor by default with levels ordered according to user input. `et()` mimics `count()` from 'dplyr'.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

LazyData true

Suggests knitr, rmarkdown

VignetteBuilder knitr

Repository <https://yea-hung.r-universe.dev>

Date/Publication 2026-05-01 18:48:56 UTC

RemoteUrl <https://github.com/yea-hung/baseverse>

RemoteRef HEAD

RemoteSha 21815b9ee0fba8dab435d2f9aacea4b00f7a2fc1

Contents

<code>bang</code>	2
<code>base_match</code>	3
<code>base_when</code>	3
<code>bracket</code>	4
<code>dollar</code>	5
<code>et</code>	5

nhanes	6
not	7
p_cor	8
p_glm	8
p_lm	9
p_t.test	10
p_table	11
p_wilcox.test	12

Index	14
--------------	-----------

bang	<i>Uses the logical not</i>
------	-----------------------------

Description

A wrapper function for !.

Usage

```
bang(x)
```

Arguments

x a logical object

Value

a logical object

Examples

```
# load the data
data(nhanes)

# check whether values of smq020 are not missing
nhanes$smq020 |> is.na() |> bang()

# check whether values of smq020 are not missing
nhanes |> dollar(smq020) |> is.na() |> bang()
```

base_match	<i>Defines factors using value-label mapping</i>
------------	--

Description

A base-R approximation of `case_match()` from 'dplyr'. Unlike `case_match()`, `base_when()` returns a factor. The levels will be ordered according to the order included in ... (see below).

Usage

```
base_match(original_variable, ..., as_factor = TRUE, string_for_na = "")
```

Arguments

<code>original_variable</code>	the original variable
<code>...</code>	the codebook, specified as a named vector, with each element in 'label'=value format, with the levels listed in the desired order
<code>as_factor</code>	logical, controlling whether the function should return a factor
<code>string_for_na</code>	string value that will be converted to NA

Value

a factor

Examples

```
# load data
data(nhanes)

# define country
nhanes<-nhanes |> transform(
  country=base_match(dmdborn4, 'USA'=1, 'Other'=2)
)
```

base_when	<i>Defines factors using logical objects</i>
-----------	--

Description

A base-R approximation of `case_when()` from 'dplyr'. Unlike `case_when()`, `base_when()` returns a factor. The levels will be ordered according to the order included in ... (see below).

Usage

```
base_when(..., as_factor = TRUE, string_for_na = "")
```

Arguments

... conditions for defining the replacement values, specified as a named list, with each element in 'label'=logical_vector format, with the levels listed in the desired order

as_factor logical, controlling whether the function should return a factor

string_for_na string value that will be converted to NA

Value

a factor

Examples

```
# load data
data(nhanes)

# define cholesterol
nhanes<-nhanes |>
  transform(
    cholesterol=base_when(
      'Desirable' = (lbxtc<200),
      'Borderline high' = (lbxtc>=200)&(lbxtc<240),
      'High' = (lbxtc>=240)
    )
  )
```

bracket

Subsets a data using bracket notation

Description

A wrapper function for bracket notation. The function only allows one index; for two-dimensional objects, the index is used on the columns.

Usage

```
bracket(data_object, index)
```

Arguments

data_object a data object

index a character, logical, or numeric vector

Value

a subset of the data

Examples

```
# load package
data(nhanes)

# look at the first few values of dmdborn4
nhanes |> bracket('dmdborn4') |> head()
```

dollar	<i>Selects a portion of a data object using dollar-sign selection</i>
--------	---

Description

A wrapper function for dollar-sign notation.

Usage

```
dollar(data_object, variable_name)
```

Arguments

data_object the data object
variable_name the variable name or element name

Value

a portion of the data object

Examples

```
# load data
data(nhanes)

# summarize lbxtc
nhanes |> dollar(lbxtc) |> summary()
```

et	<i>Tables variables</i>
----	-------------------------

Description

Creates a table for a variable, mimicking count() from 'dplyr'. ET stands for exploratory table or easy table.

Usage

```
et(data_frame, variable_name)
```

Arguments

`data_frame` the data.frame
`variable_name` the variable name

Value

a data frame

Examples

```
# load data
data(nhanes)

# define country
nhanes<-nhanes |> transform(
  country=base_match(dmdborn4, 'USA'=1, 'Other'=2)
)

# examine variable, with native piping
nhanes |> et(country)

# examine variable, with dollar-sign notation
et(nhanes$country)
```

nhanes

National Health and Nutrition Examination Survey data

Description

A subset of data from the National Health and Nutrition Examination Survey

Usage

nhanes

Format

nhanes:
A data frame with 8,153 rows and 3 columns:
seqn a unique identifier
riagendr gender
ridageyr age
dmqmiliz military status
dmdborn4 country of birth
lbxtc total cholesterol
bpxosy1 systolic blood pressure

bpxosy2 systolic blood pressure
bpxodi1 diastolic blood pressure
bpxodi2 diastolic blood pressure
mcq010 asthma
smq020 smoking status

Source

<https://wwwn.cdc.gov/nchs/nhanes/continuousnhanes/default.aspx?Cycle=2021-2023>

not	<i>Uses the logical not</i>
-----	-----------------------------

Description

A wrapper function for `!`. This is the same thing as `bang()`.

Usage

```
not(x)
```

Arguments

x a logical object

Value

a logical object

Examples

```
# load the data
data(nhanes)

# check whether values of smq020 are not missing
nhanes$smq020 |> is.na() |> not()

# check whether values of smq020 are not missing
nhanes |> dollar(smq020) |> is.na() |> not()
```

p_cor *Finds correlation coefficients, with support for piping*

Description

A wrapper function for `cor()`, with support for piping.

Usage

```
p_cor(data, x, y, ...)
```

Arguments

data	the data
x	one of the two variables
y	one of the two variables
...	additional arguments passed to <code>stats::cor()</code> .

Value

a numeric object

Examples

```
# load the data
data(nhanes)

# get correlation coefficient for systolic and diastolic
nhanes |> p_cor(bpxosy1, bpxodi1, use='complete.obs')
```

p_glm *Fits generalized linear models, with support for piping*

Description

A wrapper function for `glm()`, with the data argument listed first to support piping.

Usage

```
p_glm(data, formula, ...)
```

Arguments

data	the data
formula	the formula
...	additional arguments passed to <code>stats::glm()</code> .

Value

a glm object

Examples

```
# load the data
data(nhanes)

# define asthma
nhanes<-nhanes |> transform(
  asthma=base_match(mcq010,'No'=2,'History of asthma'=1)
)

# define smoking
nhanes<-nhanes |> transform(
  smoking=base_match(sm020,'No'=2,'History of smoking'=1)
)

# fit a model
my_model<-nhanes |> p_glm(asthma~smoking,family=binomial(link='logit'))

# obtain model details
my_model |> summary()

# obtain confidence interval
my_model |> confint() |> exp()
```

p_lm

Fits linear models, with support for piping

Description

A wrapper function for `lm()`, with the `data` argument listed first to support native piping.

Usage

```
p_lm(data, formula, ...)
```

Arguments

<code>data</code>	the data
<code>formula</code>	the formula
<code>...</code>	additional arguments passed to <code>stats::lm()</code> .

Value

an lm object

Examples

```
# load the data
data(nhanes)

# define country
nhanes<-nhanes |> transform(
  country=base_match(dmdborn4, 'USA'=1, 'Other'=2)
)

# fit a model
my_model<-nhanes |> p_lm(bpxosy1~country)

# obtain model details
my_model |> summary()

# obtain confidence interval
my_model |> confint()
```

p_t.test

Conducts t-tests, with support for piping

Description

A wrapper function for `t.test()`, with the `data` argument listed first to support piping.

Usage

```
p_t.test(data, x = NULL, y = NULL, formula = NULL, ...)
```

Arguments

<code>data</code>	the data
<code>x</code>	one of two variables
<code>y</code>	one of two variables
<code>formula</code>	a formula
<code>...</code>	additional arguments passed to <code>stats::t.test()</code> .

Value

an `htest` object

Examples

```
# load the data
data(nhanes)

# define smoking
nhanes<-nhanes |> transform(
  smoking=base_match(smq020,'No'=2,'History of smoking'=1)
)

# conduct a one-sample t-test
nhanes |> p_t.test(bpxosy1)

# conduct a two-sample t-test, using formula notation
nhanes |> p_t.test(bpxosy1~smoking)

# conduct a two-sample t-test, using formula notation
nhanes |> p_t.test(formula=bpxosy1~smoking)

# conduct a paired t-test, using x and y
nhanes |> p_t.test(bpxosy1,bpxosy2,paired=TRUE)
```

p_table

Tables variables, with support for piping

Description

A wrapper function for `table()`, with support for piping.

Usage

```
p_table(data, x, y = NULL, ...)
```

Arguments

data	the data
x	one variable
y	another variable
...	additional arguments passed to <code>table()</code> .

Value

a table object

Examples

```
# load the data
data(nhanes)

# define smoking
nhanes<-nhanes |> transform(
  smoking=base_match(smq020, 'No'=2, 'History of smoking'=1)
)

# define asthma
nhanes<-nhanes |> transform(
  asthma=base_match(mcq010, 'Asthma'=1, 'No'=2)
)

# table smoking
nhanes |> p_table(smoking)

# table smoking and asthma
nhanes |> p_table(smoking, asthma)
```

p_wilcox.test

Conducts Wilcoxon rank-sum tests, with support for piping

Description

A wrapper function for `wilcox.test()`, with the `data` argument listed first to support piping.

Usage

```
p_wilcox.test(data, x = NULL, y = NULL, formula = NULL, ...)
```

Arguments

<code>data</code>	the data
<code>x</code>	one of two variables
<code>y</code>	one of two variables
<code>formula</code>	a formula
<code>...</code>	additional arguments passed to <code>stats::wilcox.test()</code> .

Value

an `htest` object

Examples

```
# load the data
data(nhanes)

# define smoking
nhanes<-nhanes |> transform(
  smoking=base_match(smq020,'No'=2,'History of smoking'=1)
)

# conduct a one-sample wilcoxon test
nhanes |> p_wilcox.test(bpxosy1)

# conduct a two-sample wilcoxon test, using formula notation
nhanes |> p_wilcox.test(bpxosy1~smoking)

# conduct a two-sample wilcoxon test, using formula notation
nhanes |> p_wilcox.test(formula=bpxosy1~smoking)

# conduct a paired wilcoxon test, using x and y
nhanes |> p_wilcox.test(bpxosy1,bpxosy2,paired=TRUE)
```

Index

* datasets

nhanes, 6

bang, 2

base_match, 3

base_when, 3

bracket, 4

dollar, 5

et, 5

nhanes, 6

not, 7

p_cor, 8

p_glm, 8

p_lm, 9

p_t.test, 10

p_table, 11

p_wilcox.test, 12

stats::cor(), 8

stats::glm(), 8

stats::lm(), 9

stats::t.test(), 10

stats::wilcox.test(), 12

table(), 11