# Package 'na.tools'

June 25, 2018

**Title** Comprehensive Library for Working with Missing (NA) Values in Vectors

**Version** 0.3.1

**Date** 2018-06-25

**Description** This comprehensive toolkit provide a consistent and extensible framework for working with missing values in vectors. The companion package 'tidyimpute' provides similar functionality for list-like and table-like structures).
Functions exist for detection, removal, replacement, imputation, recollection, etc. of 'NAs'.

**URL** https://github.com/decisionpatterns/na.tools

**BugReports** https://github.com/decisionpatterns/na.tools/issues

**Depends** R (>= 3.1.0)

**Imports** stats, methods

**Suggests** testthat (>= 1.0.2)

**License** GPL-3 | file LICENSE

**LazyData** true

**RoxygenNote** 6.0.1.9000

**Repository** CRAN

**Encoding** UTF-8

**NeedsCompilation** no

**Author** Christopher Brown [aut, cre],
Decision Patterns [cph]

**Maintainer** Christopher Brown <chris.brown@decisionpatterns.com>

**Date/Publication** 2018-06-25 20:02:57 UTC

# R topics documented:



---

`all_na` *Tests for missing values*

---

## Description

Test if all values are missing

## Usage

```
all_na(x)

## Default S3 method:
all_na(x)

any_na(x)

is_na()

which_na(x)
```

## Arguments

| | |
|---|---|
| `x` | object to test. |

## Details

These are S3 Generics that provide default methods.

`all_na` reports if **all** values are missing.

`any_na` reports if **any** values are missing. If always returns a logical scalar.

`is_na` is a wrapper around `base::is.na()` created to keep stylistic consistenct with the other functions.

`which_na` is implemented as `which( is.na(x) )`. It is a S3 generic function.

## Value

logical scalar indicating if values are missing.

logical scalar; either TRUE or FALSE.

`integer` of indexes of `x` that corerspond to elements of `x` that are missing (`NA`). Names of the result are set to the names of `x`.

## See Also

- `base::anyNA()`
- `base::is.na()` - for the variant returning logical

## Examples

```
all_na( c( NA, NA, 1 ) )    # FALSE
all_na( c( NA, NA, NA ) )   # TRUE

df <- data.frame( char = rep(NA_character_, 3), nums=1:3)
all_na(df)  # FALSE

df <- data.frame( char = rep(NA_character_, 3), nums=rep(NA_real_,3))
all_na(df)  # TRUE

any_na( 1:10 )           # FALSE
any_na( c( 1, NA, 3 ) )  # TRUE


x <- c( 1, NA, NA, 4:6 )
which_na(x)

names(x) <- letters[1:6]
which_na(x)
```

---

`coerce_safe` *coerce_safe*

---

## Description

Coerce values in a safe, non-destructive and consistent way.

## Usage

```
coerce_safe(object, class, alert = stop, ..., alert_irreversible = alert,
  alert_na = alert)
```

## Arguments

| | |
|---|---|
| `object` | to be coerced |
| `class` | character; class to which `object` should be coerced. |
| `alert` | function to use to raise exceptions: (Default: `base::stop()`) |
| `...` | unused |
| `alert_irreversible` | function to raise alert when coercion is not reversible. See Details. |
| `alert_na` | function to raise when `NA`s are produced. |

`coerce_safe` transform the `object` to `class` in a safe, consistent, non-destructive way.

Safe means that coercison:

1. is non-destructive (i.e information is not lost in the transformation )
2. is reversible:
$$f^{-1}(f(x)) == x$$
3. does not introduce (additional) missing values (`NA`)

By default, `corece_safe` raises an alert (message|warning|error) when the attempted coercion violates these constraints. The `alert` argument (and `alert_irreversible` or `alert_na`) can be used to fleixble change the response. Valid values for these are `base::message()`, `base::warning()` and base::stop among others.

## Value

`object` coerced to `class` but ensured that there has been no loss in data and no additional Missonve values introduced.

## Note

There must be a `as` method to the reverse coercion for this function to work.

## See Also

methods::as ‘coercion::try_as()“

## Examples

```
## Not run:
  # Error
  coerce_safe(1.01, "integer")  # 1.01 != 1
  coerce_safe( c("1","2","a"), "integer" )

## End(Not run)
```

`impute-commutative` *Imputation by Cummutative Functions Impute using replacement values calculated from a univariate, cummuative function.*

## Description

Imputation by Cummutative Functions

Impute using replacement values calculated from a univariate, cummuative function.

`na.median` imputes with the median value of `x`. The median is only valid for numeric or logical values.

## Usage

```
na.max(.x, ...)

na.min(.x, ...)

na.mean(.x, ...)

na.median(.x, ...)

na.quantile(.x, ...)

na.mode(.x, ...)

na.most_freq(.x, ...)
```

## Arguments

| | |
|---|---|
| `.x` | vector in which `NA` values are to be replaced. The ordering of `x` does not matter. |
| `...` | additional arguments passed to lower-level summary functions. |

## Details

This collection of functions calculates a replacement value using an unvariate function where the order of values in `x` do not matter, i.e. commutative.

`na.max` and `na.min` replace missing values (`NA`) with the maximum or minimum *of non-missing values* `x`. ( Internally: `base::max(..., na.rm=TRUE)` and `base::min(..., na.rm=TRUE)`. `...` has no affect.

`na.mean` replaces `NA` values with the mean of `x`. Internally, `mean(x, na.rm=TRUE, ... )` is used. If mean cannot be calculated (e.g. x isn't numeric) then `x` is returned with a warning.

`na.quantile` imputes with a quantile. The quantile is specified by a `probs` argument that is passed to `stats::quantile()`. If `probs` can be a scalar value in which all values are replaced by that quantile or a vector of `length(.x)` values which replaces the missing values of x with the `probs`. The ability to provide a vector may be deprecated in the future.

`na.mode` replaces all `NA` with the most frequently occuring value. In the event of ties, the value encounter first in `.x` is used.

`na.most_freq` is an alias for `na.mode`.

## Value

A vector of `class(x)` and `length(x)` in which missing values (NA) have been replaced the result of a function call:

$$fun(x, ...)$$

## See Also

- `na.replace()` - used internally by these functions
- `na.constant()`
- `base::max()` and `base::min()`

`median()`

- `quantile()`

## Examples

```
na.median( c(1,2,NA_real_,3) )

na.quantile( c(1,2,NA_real_,3), prob=0.4 )

na.mode( c(1,1,NA,4) )
na.mode( c(1,1,4,4,NA) )
```

---

`impute-constant` *Impute by Constant Value Replaces* `NA`*s by a constant*

---

## Description

Impute by Constant Value

Replaces `NA`s by a constant

## Usage

```
na.constant(.x, .na)

na.inf(.x)

na.neginf(.x)

na.true(.x)
```

```
na.false(.x)

na.zero(.x)
```

## Arguments

| | |
|---|---|
| .x | vector; of values to have the NA |
| .na | scalar to use as replacement. |

## Details

These functions replace **ALL** NA values in x with an scalar value specified by .na.

na.constant replaces missing values with a scalar constant. It is a wrapper around na.replace() but permits .na to only be a scalar.

na.inf and na.neginf replace all missing values with Inf and -Inf repectively. ‘.

na.true and na.false replace missing values with TRUE and FALSE respectively.

na.zero replaces missing values with 0 which gets coerced to the class(x) as needed.

## Value

A vector with the type and length of x with all missing values replaces by .na.

## See Also

- na.replace() the underlying function that performs the replacement.

## Examples

```
na.constant( c(1,NA,2), -1 )

na.inf( c( 1, 2, NA, 4) )
na.neginf( c( 1, 2, NA, 4) )

na.true( c(TRUE, NA_logical, FALSE) )   # T T F
na.false( c(TRUE, NA_logical, FALSE) )  # T F F



na.zero( c(1,NA,3) )  # 1 0 3
```

`impute-noncommutative` *non-commutative imputation Impute missing values using non-commutative functions, i.e. where the order* **matters**.

## Description

non-commutative imputation

Impute missing values using non-commutative functions, i.e. where the order **matters**.

## Usage

```
na.cummax(.x, ...)

na.cummin(.x, ...)

na.cumsum(.x, ...)

na.cumprod(.x, ...)
```

## Arguments

| | |
|---|---|
| `.x` | atomic-vector with 0 or more missing values |
| `...` | additional arguments |

## Details

Non-commutative imputations functions assume that `.x` is in the proper order since the values depend on order. Usually, this is relevant then `.x` is part of a table.

These functions replaces `NA` values with the cummulative max of `.x`. Internally, `fun(.x, na.rm=TRUE, ... )` is used. If the function cannot be calculated (e.g. `.x` isn't numeric) then `x` is returned unchanged with a warning.

Use of `na.cumsum` and `na.cumprod` are dangerous since they omit missing values that may contribute to

## See Also

- `base::cummax()`
- impute-commutative

na.bootstrap *na.bootstrap*

## Description

Replace missing values with value randomly drawn from x

## Usage

```
na.bootstrap(.x, ...)

na.resample(.x, ...)
```

## Arguments

| | |
|---|---|
| .x | vector with |
| ... | additional arguments passed to base::sample() |

## Details

na.random replaces missing values by sampling the non-missing values. By default aampling occurs **with replacement** since more valuables may be needed than are available. This function is based on base::sample().

The default is to replace bv sampling a population defined by the non-missing values of .x **with replacement**

na.random is an alias for na.bootstrap. ‘

## Note

na.bootstrap is **non-deterministic**. Use base::set.seed() to make it deterministic

## See Also

- base::sample()

## Examples

```
x <- c(1,NA,3)
na.bootstrap(x)
```

`na.replace` *Replace Missing Values*

## Description

Replaces NA values with explicit values.

## Usage

```
na.replace(x, .na, ...)

na.explicit(x)
```

## Arguments

| | |
|---|---|
| x | vector in which NA values are to be replaced. |
| .na | scalar, length(x)-vector or function used to replace NA. See #Details. |
| ... | additional arguments passed to .na when it is a function. |

## Details

`na.replace` replaces missing values in `x` by `.na` if possible.

In R, replacement of values can cause a change in the class/type of an object. This is not often desired. `na.replace` is class/type-safe and length-safe. It replaces missing values without changing the `x`'s class or length regardless of the value provided by `.na`.

**Param:** `x`

If `x` is **categorical** (e.g. character or factor), `.na` is optional. The default is "(NA)" and can be set with `options( NA_explicit_ = new_value )`. It can also be referenced directly with NA_explicit_.

If `x` is a **factor**, unique values of `.na` not in already present in `levels(x)` will be added. They are appended silently unless `getOption('verbose')==TRUE` in which a message reports the added levels.

**Param:** `.na`

`.na` can be either a scalar, vector or function.

If a **scalar**, each missing value of `x` is replaced by `na`.

If a **vector**, `.na` must have length(x). `Missing values ofxare replaced by corresponding elements of.na. Recycli`

If a **function**, `x` is transformed by .na‘ with:

```
.na(x, ...)
```

then preceding with normal operations.

`na.explicit` is an alias for na.replace that uses NA_explicit_ for ‘.na“; it returns x unchanged if it cannot change the value.

## Value

A vector with the class and length of x. NAs in x will be replaced by .na. .na is coerced as necessary.

## See Also

- base::ifelse(), base::replace()
- forcats::fct_explicit_na - which only handles factors

## Examples

```
  # Integers and numerics
  na.replace( c(1,NA,3,NA), 2 )    # 1 2 3 2
  na.replace( c(1,NA,3,NA), 1:4 )  # 1 2 3 4

  # This produces an error because it would change the type
  ## Not run:
  na.replace( c(1,NA,3,NA), letters[1:4] )  # "1" "b" "3" "d"

## End(Not run)

  # Characters
  lets <- letters[1:5]
  lets[ c(2,4) ] <- NA
  na.replace(lets)  # replace with NA_explicit_

  # Factors
  fct <- as.factor( c( NA, letters[2:4], NA) )
  fct
  na.replace(fct, "z")  # z b c d z  -- level z added
  na.replace(fct, letters[1:5] )
  na.replace(fct)

 ## Not run:
   na.replace( rep(NA,3), rep(NA,3) )

## End(Not run)
```

---

na.rm *na.rm*

---

## Description

Removes NA values from objects

## Usage

```
na.rm(object, ...)
```

### Arguments

| | |
|---|---|
| object | to remove NAs from |
| ... | further arguments special methods could require. |

### Details

For **vectors** this is the same as stats::na.omit() or stats::na.exclude(). It will also work on recursive objects.

This is predominantly maintained for syntactic convenience since a number of functions have na.omir

### Value

An object of the same class with all NA values removed. For data.frame and data.table objects entire columns are removed if they contain solely NA values.

### See Also

- stats::na.omit(), stats::na.exclude()
- all_na()

---

## na.unreplace *na.unreplace*

---

### Description

Change values to NAs, ie make explicit NAs back to NA

### Usage

```
na.unreplace(x, values)

## Default S3 method:
na.unreplace(x, values = NULL)

## S3 method for class 'character'
na.unreplace(x, values = c("NA", NA_explicit_))

## S3 method for class 'factor'
na.unreplace(x, values = c("NA", NA_explicit_))

na.implicit(x, values)
```

### Arguments

| | |
|---|---|
| x | object |
| values | values that are (or can be coerced to) class(x) that are to be set to NA. |

## Details

na.unreplace replaces values by NA. It is meant to be nearly inverse operation to na_replace (and na_explicit). It can be used on both atomic and recursive objects. Unlike na.replace however, values express the values that if matched are set to NA. It is basically:

```
x[ x
```

na.unreplace is a S3 method that can be used to defince additional methods for other objects.

## See Also

- na.replace()

## Examples

```
na.unreplace( c(1,2,3,4), 3 )
na.unreplace( c("A", "(NA)", "B", "C") )
na.unreplace( c("A", NA_explicit_, "B", "C") )

df <- data.frame( char=c('A', 'NA', 'C', NA_explicit_), num=1:4 )
na.unreplace(df)
```

---

NA_explicit_ *NA_explicit_*

---

## Description

Default replacement for missing values in categorical vectors.

## Usage

```
NA_explicit_
```

## Format

An object of class character of length 1.

## Details

NA_explicit_ is used as a default replacement for categorical vectors.

It is an active binding to getOptions('NA_explicit_') and is exported to the callers namespace.

To change the value of NA_explicit use:

```
options( NA_explicit = new_value )
```

NA_explicit_ cannot be directly set.

## See Also

na.replace()

---

NA_logical *NA_logical*

---

## Description

NA_logical

## Usage

```
NA_logical
```

## Format

An object of class `logical` of length 1.

## Details

This simply creates a NA_logical variable. This is the same as NA

---

n_na *Counts how many values are NA*

---

## Description

Returns the number of values that are NA

## Usage

```
n_na(x)

na.howmany(x)

na.n(x)

pct_na(x)

na.pct(x)
```

## Arguments

| | |
|---|---|
| x | object to count how many values are NA |

## Details

`n_na` counts the number of missing values. `na.n` is an alias in the dplyr style.

`pct_na` gives the percentage of values that are `NA`

## Value

`n_na` returns an integer. `pct_na` returns a numeric value 0-1.

## Examples

```
x <- c( 1, NA, NA, 4:5 )
n_na(x)
pct_na(x)
```

# Index