# Package 'DTwrappers'

May 7, 2026

**Title** Simplified Data Analysis with Wrapper Functions for the 'Data.Table' Package

**Version** 0.0.2

**Depends** R (>= 3.1.0)

**Description** Provides functionality for users who are learning R or the techniques of data analysis. Written as a collection of wrapper functions, the 'DTwrapper' package facilitates many core operations of data processing. This is achieved with relatively few requirements about the order of the processing steps or knowledge of specialized syntax. 'DTwrappers' creates coding results along with translations to data.table's code. This enables users to benefit from the speed and efficiency of data.table's calculations. Furthermore, the package also provides the translated code for educational purposes so that users can review working examples of coding syntax and calculations.

**License** GPL-3

**Encoding** UTF-8

**RoxygenNote** 7.1.1

**Suggests** knitr, rmarkdown, covr, testthat (>= 2.1.0)

**VignetteBuilder** knitr

**Imports** data.table

**NeedsCompilation** no

**Author** David Shilane [aut], Mayur Bansal [ctb, cre]

**Maintainer** Mayur Bansal <mb4511@columbia.edu>

**Repository** CRAN

**Date/Publication** 2021-06-21 06:20:08 UTC

## Contents

---

`add.backtick` *Add backtick*

---

## Description

Function that add backticks to the input variables.

## Usage

```
add.backtick(x, include.backtick = "as.needed", dat = NULL)
```

## Arguments

| | |
|---|---|
| `x` | Character value specifying the name of input parameters. |
| `include.backtick` | specifies whether a backtick should be added. Parameter values should be either 'all' or 'as.needed' |
| `dat` | the dataset |

## Value

None

---

`dt.calculate` *dt.calculate*

---

## Description

This function allows a user to apply one or more functions to all of the specified variables in a data.frame or data.table object. It is built as a wrapper function of data.table's method of applying functions to variables while allowing for filtering and grouping steps. This allows a user to easily calculate many results, e.g. the.functions = c("mean", "median", "sd") on multiple columns, e.g. the.variables = c("Age", "Income") while also filtering and grouping the data. Options also exist to return a data.table coding statement (result = "code") for educational purposes or both the result and the code together (result = "all"). For examples, please see the vignette.

## Usage

```
dt.calculate(
  dt.name,
  the.functions,
  the.variables = ".",
  the.filter = NULL,
  grouping.variables = NULL,
  sortby.group = TRUE,
  other.params = "",
  table.format = "long",
  add.function.name = TRUE,
  individual.variables = TRUE,
  output.as.table = TRUE,
  return.as = "result",
  envir = .GlobalEnv,
  ...
)
```

## Arguments

| | |
|---|---|
| dt.name | a character value specifying the name of a data.frame or data.table object to select data from. A variable called dat should be referred to with dt.name = "dat" when using the function. |
| the.functions | A character vector specifying the name of the functions to apply to the.variables. Each function included in the.functions will be separately applied to each variable in the.variables. |
| the.variables | A character or numeric vector specifying the variables to perform the calculations on. For character vectors, only values that exist in the names of the data will be used. For numeric vectors, only the values of unique(floor(sorting.variables)) that are in 1:ncol() of your data will be used. Then these indices will be used to select column names from the data. Other values in sorting.variables that do not correspond to a defined column will be excluded from the calculation. When the.variables includes ".", then all values in names(dat) will be selected. Values of the.variables that also exist in grouping.variables will be excluded from the.variables (but grouped by these values). |
| the.filter | a character value, numeric vector, logical vector, or expression stating the logical operations used to filter the data. The filtering step will be applied prior to generating the counts. Defaults to NULL unless otherwise specified. Logical vectors will be converted to a numeric filter, e.g. c(TRUE, TRUE, FALSE) will become 1:2 to signify which rows should be selected. |
| grouping.variables | A character or numeric vector specifying the variables to perform the calculations on. For character vectors, the values may be either column names of the data or calculations based upon them (see the vignette for examples). For numeric vectors, only the values of unique(floor(grouping.variables)) that are in 1:ncol() of your data will be used. Then these indices will be mapped to the corresponding column names from the data. When NULL, no grouping will be performed. |

`sortby.group` A logical value specifying whether the grouping should be sorted (TRUE, the default value) or as is (FALSE).

`other.params` A character value specifying any additional parameters needed to call the.functions. For instance, if the.functions = "mean", and you would like to remove missing values, then specifying other.params = "na.rm = TRUE" as a character would suffice. Multiple parameters can be specified with comma separation, e.g. other.params = "trim = 1, na.rm = TRUE". Note that all of the parameters supplied must apply to all of the.functions

`table.format` specify the format of the table depending on the desired output i.e. "long" or "wide"

`add.function.name` A logical value specifying whether the name of the function applied should be appended to the column names in the resulting table. Only applies if the.functions is of length 1.

`individual.variables` a logical variable specifying if variables are grouped or individual

`output.as.table` a logical variable to specify if output should be a table or not

`return.as` a character value specifying what output should be returned. return.as = "result" provides the table of counts. return.as = "code" provides a data.table coding statement that can generate the table of counts. return.as = "all" provides a list containing both the resulting table and the code.

`envir` the environment in which the code would be evaluated; .GlobalEnv by default.

`...` additional arguments to be passed

## Value

Depending on the value of return.as, the output will be a) a character value (return.as = 'code'), b) a coding output, typically a data.table (return.as = 'result'), or c) a list containing both the code and output (return.as = 'all')

## Source

DTwrappers::create.dt.statement

DTwrappers::eval.dt.statement

DTwrappers::add.backtick

## Examples

```
n <- nrow(iris)
dat <- data.table::as.data.table(x = iris[sample(x = 1:n, size = n, replace = FALSE),])
dt.calculate(dt.name = "dat", the.variables = c("Sepal.Length"),
the.functions = c("mean", "sd"), return.as = "all")
```

dt.choose.cols *dt.choose.cols*

## Description

This function selects columns from a data.frame or data.table. It is built as a wrapper function of data.table's selection step (using .SD in the j step while specifying the .SDcols argument). Selections may also be supplied to limit the rows to consider, with options for the first or last k rows or a subset based upon a vector like c(3:5, 9:10). Filtering of the rows (e.g. Age < 50) may also be applied using the.filter. Grouped operations may be used to make these selections of columns and rows in each category. Options also exist to return a data.table coding statement (result = "code") for educational purposes or both the result and the code together (result = "all"). For examples, please see the vignette.

## Usage

```
dt.choose.cols(
  dt.name,
  the.variables = ".",
  the.filter = NULL,
  grouping.variables = NULL,
  sortby.group = TRUE,
  first.k = NULL,
  last.k = NULL,
  row.indices = NULL,
  return.as = "result",
  envir = .GlobalEnv
)
```

## Arguments

| | |
|---|---|
| dt.name | a character value specifying the name of a data.frame or data.table object to select data from. A variable called dat should be referred to with dt.name = "dat" when using the function. |
| the.variables | A character or numeric vector specifying the variables that we want to select. For character vectors, only values that exist in the names of the data will be used. For numeric vectors, only the values of unique(floor(sorting.variables)) that are in 1:ncol() of your data will be used. Then these indices will be used to select column names from the data. Only values that exist in the names of the data will be used; other values in the.variables will be excluded from the calculation. When the.variables includes ".", then all of the variables will be selected. Values of the.variables that also exist in grouping.variables will be excluded from the.variables (but grouped by these values). |
| the.filter | a character value, numeric vector, logical vector, or expression stating the logical operations used to filter the data. The filtering step will be applied prior to generating the counts. Defaults to NULL unless otherwise specified. Character |

values such as 'Age < 50' or 'c(1:3, 7:10)' may be used. Numeric vectors such as c(1:3, 7:10) that specify the row indices may be used. Logical vectors will be converted to a numeric filter, e.g. c(TRUE, TRUE, FALSE) will become 1:2 to signify which rows should be selected. Expressions may be used to specify a logical operation such as expression(Age < 50) as well. Defaults to NULL to indicate that no filtering of the data should be applied.

`grouping.variables`
: A character or numeric vector specifying the variables to perform the calculations on. For character vectors, the values may be either column names of the data or calculations based upon them (see the vignette for examples). For numeric vectors, only the values of unique(floor(grouping.variables)) that are in 1:ncol() of your data will be used. Then these indices will be mapped to the corresponding column names from the data. When NULL, no grouping will be performed.

`sortby.group`
: A character value specifying whether the grouping should be sorted (keyby) or as is (by). Defaults to keyby unless "by" is specified.

`first.k`
: An integer indicating how many rows to select starting from the first row. Note that grouping statements will select up to this number of rows in each group. Additionally, if first.k is larger than the number of records in a group, then the maximum number of records will be selected. When non-integer or non-positive values of first.k are selected, the algorithm will select first.k = max(c(1, round(first.k))). If first.k is not a numeric or integer value, then by default first.k is set to select all of the rows. Specifying row.indices takes precedence to specifying the parameter first.k; if row.indices is not NULL, then row.indices will be used, and first.k will not. Meanwhile, first.k takes precedence to last.k when both are specified. See below.

`last.k`
: An integer indicating how many rows to select starting from the last row. Note that grouping statements will select up to this number of rows in each group. Additionally, if last.k is larger than the number of records in a group, then the maximum number of records will be selected. When non-integer or non-positive values of last.k are selected, the algorithm will select last.k = max(c(1, round(last.k))). If last.k is not a numeric or integer value, then by default last.k is set to select all of the rows. Specifying row.indices takes precedence to specifying the parameter last.k (see below); if row.indices is not NULL, then it will be used, and last.k will not. Meanwhile, first.k takes precedence to last.k when both are specified.

`row.indices`
: An integer vector specifying the row indices to return. When grouping.variables is specified, these indices will be applied to each group. Note that specifications outside of the range from 1 to the number of rows will be limited to existing rows from the data and group. Specifying row.indices takes precedence to specifying the parameters first.k and last.k. If row.indices is not NULL, it will be used.

`return.as`
: a character value specifying what output should be returned. return.as = "result" provides the table of counts. return.as = "code" provides a data.table coding statement that can generate the table of counts. return.as = "all" provides a list containing both the resulting table and the code.

`envir`
: the environment in which the code would be evaluated; .GlobalEnv by default.

### Value

Depending on the value of return.as, the output will be a) a character value (return.as = 'code'), b) a coding output, typically a data.table (return.as = 'result'), or c) a list containing both the code and output (return.as = 'all')

### Source

DTwrappers::create.dt.statement

DTwrappers::eval.dt.statement

---

`dt.choose.rows` *dt.choose.rows*

---

### Description

This function filters the rows of a data.table or data.frame object. It is built as a wrapper function of data.table's filtering method (the i step). A series of logical tests on variables within the data may be specified. Options also exist to return a data.table coding statement (result = "code") for educational purposes or both the result and the code together (result = "all"). For examples, please see the vignette.

### Usage

```
dt.choose.rows(
  dt.name,
  the.filter = NULL,
  return.as = "result",
  envir = .GlobalEnv
)
```

### Arguments

| | |
|---|---|
| `dt.name` | a character value specifying the name of a data.frame or data.table object to select data from. A variable called dat should be referred to with dt.name = "dat" when using the function. |
| `the.filter` | a character value, numeric vector, logical vector, or expression stating the logical operations used to filter the data. The filtering step will be applied prior to generating the counts. Defaults to NULL unless otherwise specified. Character values such as 'Age < 50' or 'c(1:3, 7:10)' may be used. Numeric vectors such as c(1:3, 7:10) that specify the row indices may be used. Logical vectors will be converted to a numeric filter, e.g. c(TRUE, TRUE, FALSE) will become 1:2 to signify which rows should be selected. Expressions may be used to specify a logical operation such as expression(Age < 50) as well. Defaults to NULL to indicate that no filtering of the data should be applied. |

| | |
|---|---|
| return.as | a character value specifying what output should be returned. return.as = "result" provides the table of counts. return.as = "code" provides a data.table coding statement that can generate the table of counts. return.as = "all" provides both the resulting table and the code. If the coding statement was specified using calls to get() or eval(), then both an original.statement and the resulting code (a translated statement from the getDTeval package) will be provided. |
| envir | the environment in which the code would be evaluated; .GlobalEnv by default. |

## Value

Depending on the value of return.as, the output will be a) a character value (return.as = 'code'), b) a coding output, typically a data.table (return.as = 'result'), or c) a list containing both the code and output (return.as = 'all')

## Note

the data.frame dat will be converted to a data.table object to facilitate efficient counting of the rows.

## Source

DTwrappers::create.dt.statement

DTwrappers::eval.dt.statement

## Examples

```
n <- nrow(iris)
dat <- data.table::as.data.table(x = iris[sample(x = 1:n, size = n, replace = FALSE),])
dt.count.rows(dt.name = "dat", count.name = "Total Rows", return.as = "all")
```

---

# dt.count.rows *dt.count.rows*

---

## Description

This function counts the number of qualifying rows in a data.table or data.frame object. It is built as a wrapper function of data.table's filter (the i step). These counts may be comprehensive for the entire table or conducted in groups. The full data can also be filtered for qualifying cases prior to conducting the counts. This function returns a data.table object that shows the counts in one column along with additional columns for any specified grouping variables. Options also exist to return a data.table coding statement (result = "code") for educational purposes or both the result and the code together (result = "all"). For examples, please see the vignette.

## Usage

```
dt.count.rows(
  dt.name,
  the.filter = NULL,
  grouping.variables = NULL,
  sortby.group = TRUE,
  count.name = "N",
  return.as = "result",
  envir = .GlobalEnv
)
```

## Arguments

| | |
|---|---|
| dt.name | a character value specifying the name of a data.frame or data.table object to select data from. A variable called dat should be referred to with dt.name = "dat" when using the function. |
| the.filter | a character value, numeric vector, logical vector, or expression stating the logical operations used to filter the data. The filtering step will be applied prior to generating the counts. Defaults to NULL unless otherwise specified. Character values such as 'Age < 50' or 'c(1:3, 7:10)' may be used. Numeric vectors such as c(1:3, 7:10) that specify the row indices may be used. Logical vectors will be converted to a numeric filter, e.g. c(TRUE, TRUE, FALSE) will become 1:2 to signify which rows should be selected. Expressions may be used to specify a logical operation such as expression(Age < 50) as well. Defaults to NULL to indicate that no filtering of the data should be applied. |
| grouping.variables | A character or numeric vector specifying the variables to perform the calculations on. For character vectors, the values may be either column names of the data or calculations based upon them (see the vignette for examples). For numeric vectors, only the values of unique(floor(grouping.variables)) that are in 1:ncol() of your data will be used. Then these indices will be mapped to the corresponding column names from the data. When NULL, no grouping will be performed. |
| sortby.group | a character value specifying whether the table of counts should be sorted by group ("sorted") or as is (any other selected value). Defaults to "sorted". |
| count.name | a character value specifying the name of the column of counts in the resulting table. This value defaults to "N" unless otherwise specified. |
| return.as | a character value specifying what output should be returned. return.as = "result" provides the table of counts. return.as = "code" provides a data.table coding statement that can generate the table of counts. return.as = "all" provides both the resulting table and the code. If the coding statement was specified using calls to get() or eval(), then both an original.statement and the resulting code (a translated statement from the getDTeval package) will be provided. |
| envir | the environment in which the code would be evaluated; .GlobalEnv by default. |

## Value

Depending on the value of return.as, the output will be a) a character value (return.as = 'code'), b) a coding output, typically a data.table (return.as = 'result'), or c) a list containing both the code and output (return.as = 'all')

## Note

the data.frame dat will be converted to a data.table object to facilitate efficient selection.

## Source

DTwrappers::create.dt.statement

DTwrappers::eval.dt.statement

## Examples

```
n <- nrow(iris)
dat <- data.table::as.data.table(x = iris[sample(x = 1:n, size = n, replace = FALSE),])
dt.count.rows(dt.name = "dat", return.as = "all")
```

---

`dt.define.variable` *dt.define.variable*

---

## Description

This method allows a user to add a new variable to an existing data.frame or data.table. It can also be used to update previously defined variables. It is built as a wrapper function of data.table's method of defining new variables by reference. The new values can be stated either through a statement of the calculation or by directly providing a vector of values. These updates can also be performed on a subset of the data by incorporating a filter. Options also exist to return a data.table coding statement (result = "code") for educational purposes or both the result and the code together (result = "all"). For examples, please see the vignette.

## Usage

```
dt.define.variable(
  dt.name,
  variable.name,
  the.values,
  specification = "by.expression",
  the.filter = NULL,
  grouping.variables = NULL,
  sortby.group = TRUE,
  return.as = "result",
  envir = .GlobalEnv,
  ...
)
```

## Arguments

| | |
|---|---|
| dt.name | a character value specifying the name of a data.frame or data.table object to select data from. A variable called dat should be referred to with dt.name = "dat" when using the function. |
| variable.name | a character value specifying the name of the new column. |
| the.values | a vector or character value. When specified as a vector, this should contain the values of the new column. When specified as a character value, it should include a functional form that specifies how to calculate the new values. See the specification parameter for more details. |
| specification | A character value. When specification = "by.value", the new variable will be defined in terms of the vector the.values. Otherwise the new variable is specified in a functional form, e.g. the.values = "rnorm(n = 3)". |
| the.filter | a character value, logical vector, or expression stating the logical operations used to filter the data. See create.filter.expression for details. The filtering step will be applied prior to generating the counts. Defaults to NULL unless otherwise specified. |
| grouping.variables | A character or numeric vector specifying the variables to perform the calculations on. For character vectors, the values may be either column names of the data or calculations based upon them (see the vignette for examples). For numeric vectors, only the values of unique(floor(grouping.variables)) that are in 1:ncol() of your data will be used. Then these indices will be mapped to the corresponding column names from the data. When NULL, no grouping will be performed. |
| sortby.group | A logical value specifying whether the grouping should be sorted (TRUE, the default value) or as is (FALSE). |
| return.as | a character value specifying what output should be returned. return.as = "result" provides the table of counts. return.as = "code" provides a data.table coding statement that can generate the table of counts. return.as = "all" provides both the resulting table and the code. |
| envir | the environment in which the code would be evaluated; .GlobalEnv by default. |
| ... | other additional arguments if needed |

## Value

Depending on the value of return.as, the output will be a) a character value (return.as = 'code'), b) a coding output, typically a data.table (return.as = 'result'), or c) a list containing both the code and output (return.as = 'all')

## Note

the data.frame dat will be converted to a data.table object to facilitate adding the new column by reference (e.g. efficiently with regard to the usage of memory)

## Source

DTwrappers::create.dt.statement

DTwrappers::eval.dt.statement

---

`dt.first.k.rows` *dt.first.k.rows*

---

## Description

This function returns the first k rows from the given data. It is built as a wrapper function of data.table's filter (the i step). This calculation can be specified either overall or in groups. A filter can also be applied so that only qualifying values would be considered. A subset of the variables may also be selected. Options also exist to return a data.table coding statement (result = "code") for educational purposes or both the result and the code together (result = "all"). For examples, please see the vignette.

#' @param dt.name a character value specifying the name of a data.frame or data.table object to select data from.

## Usage

```
dt.first.k.rows(
  dt.name,
  k = NULL,
  the.variables = ".",
  the.filter = NULL,
  grouping.variables = NULL,
  sortby.group = TRUE,
  return.as = "result",
  envir = .GlobalEnv,
  ...
)
```

## Arguments

| | |
|---|---|
| `dt.name` | a character value specifying the name of a data.frame or data.table object to select data from. A variable called dat should be referred to with dt.name = "dat" when using the function. |
| `k` | A numeric variable specifying the number of rows to select |
| `the.variables` | A character or numeric vector specifying the variables to perform the calculations on. For character vectors, only values that exist in the names of the data will be used. For numeric vectors, only the values of unique(floor(sorting.variables)) that are in 1:ncol() of your data will be used. Then these indices will be used to select column names from the data. Other values in sorting.variables that do not correspond to a defined column will be excluded from the calculation. When the.variables includes ".", then all values in names(dat) will be selected. Values of the.variables that also exist in grouping.variables will be excluded from the.variables (but grouped by these values). |

`the.filter` a character value, logical vector, or expression stating the logical operations used to filter the data. See create.filter.expression for details. The filtering step will be applied prior to generating the counts. Defaults to NULL unless otherwise specified.

`grouping.variables` a character vector specifying the variables to group by in the calculation. Only variables in the data will be used. When NULL, no grouping will be performed.

`sortby.group` A logical value specifying whether the grouping should be sorted (TRUE, the default value) or as is (FALSE).

`return.as` a character value specifying what output should be returned. return.as = "result" provides the resulting table. return.as = "code" provides a data.table coding statement that can generate the resulting table. return.as = "all" provides both the resulting table and the code. If the coding statement was specified using calls to get() or eval(), then both an original.statement and the resulting code (a translated statement from the getDTeval package) will be provided.

`envir` the environment in which the code would be evaluated; .GlobalEnv by default.

`...` additional arguments to be passed

## Value

Depending on the value of return.as, the output will be a) a character value (return.as = 'code'), b) a coding output, typically a data.table (return.as = 'result'), or c) a list containing both the code and output (return.as = 'all')

## Note

Calls dt.choose.cols.R with first.k = k.

## Source

DTwrappers::dt.choose.cols

## Examples

```
n <- nrow(iris)
dat <- data.table::as.data.table(x = iris[sample(x = 1:n, size = n, replace = FALSE),])
dt.first.k.rows(dt.name = "dat", k = 2, the.variables = c("Sepal.Length", "Sepal.Width"),
grouping.variables = "Species", return.as = "all")
```

`dt.last.k.rows` *dt.last.k.rows*

## Description

This function returns the last k rows from the given data. It is built as a wrapper function of data.table's filter (the i step). This calculation can be specified either overall or in groups. A filter can also be applied so that only qualifying values would be considered. A subset of the variables may also be selected. Options also exist to return a data.table coding statement (result = "code") for educational purposes or both the result and the code together (result = "all"). For examples, please see the vignette.

## Usage

```
dt.last.k.rows(
  dt.name,
  k = NULL,
  the.variables = ".",
  the.filter = NULL,
  grouping.variables = NULL,
  sortby.group = TRUE,
  return.as = "result",
  envir = .GlobalEnv,
  ...
)
```

## Arguments

| | |
|---|---|
| `dt.name` | a character value specifying the name of a data.frame or data.table object to select data from. |
| `k` | A numeric variable specifying the number of rows to select |
| `the.variables` | A character or numeric vector specifying the variables to perform the calculations on. For character vectors, only values that exist in the names of the data will be used. For numeric vectors, only the values of unique(floor(sorting.variables)) that are in 1:ncol() of your data will be used. Then these indices will be used to select column names from the data. Other values in sorting.variables that do not correspond to a defined column will be excluded from the calculation. When the.variables includes ".", then all values in names(dat) will be selected. Values of the.variables that also exist in grouping.variables will be excluded from the.variables (but grouped by these values). |
| `the.filter` | a character value, logical vector, or expression stating the logical operations used to filter the data. See create.filter.expression for details. The filtering step will be applied prior to generating the counts. Defaults to NULL unless otherwise specified. |
| `grouping.variables` | a character vector specifying the variables to group by in the calculation. Only variables in the data will be used. When NULL, no grouping will be performed. |

| | |
|---|---|
| sortby.group | A logical value specifying whether the grouping should be sorted (TRUE, the default value) or as is (FALSE). |
| return.as | a character value specifying what output should be returned. return.as = "result" provides the resulting table. return.as = "code" provides a data.table coding statement that can generate the resulting table. return.as = "all" provides both the resulting table and the code. If the coding statement was specified using calls to get() or eval(), then both an original.statement and the resulting code (a translated statement from the getDTeval package) will be provided. |
| envir | the environment in which the code would be evaluated; .GlobalEnv by default. |
| ... | additional arguments to be passed |

## Value

Depending on the value of return.as, the output will be a) a character value (return.as = 'code'), b) a coding output, typically a data.table (return.as = 'result'), or c) a list containing both the code and output (return.as = 'all')

## Note

Calls dt.choose.cols.R with last.k = k.

## Source

DTwrappers::dt.choose.cols

## Examples

```
n <- nrow(iris)
dat <- data.table::as.data.table(x = iris[sample(x = 1:n, size = n, replace = FALSE),])

dt.last.k.rows(dt.name = "dat", k = 2, the.variables = c("Sepal.Width"),
grouping.variables = "Species", return.as = "all")
```

---

# dt.remove.variables *dt.remove.variables*

---

## Description

A function to remove selected columns from a data.frame or data.table object.

## Usage

```
dt.remove.variables(
  dt.name,
  the.variables,
  return.as = "result",
  envir = .GlobalEnv,
  ...
)
```

### Arguments

| | |
|---|---|
| dt.name | a character value specifying the name of a data.frame or data.table object to select data from. A variable called dat should be referred to with dt.name = "dat" when using the function. |
| the.variables | A character or numeric vector specifying the variables that we want to remove. For character vectors, only values that exist in the names of the data will be used. For numeric vectors, only the values of unique(floor(sorting.variables)) that are in 1:ncol() of your data will be used. Then these indices will be used to select column names from the data. |
| return.as | a character value specifying what output should be returned. return.as = "result" provides the updated data. return.as = "code" provides a data.table coding statement. return.as = "all" provides a list object including both the resulting output and the code. |
| envir | a specification of the environment in which the data (referenced by dt.name) exists, with the global environment as the default value.#' |
| ... | additional arguments if required |

### Value

a 'data.table' object.

### Source

DTwrappers::create.dt.statement

DTwrappers::eval.dt.statement

### Examples

```
n <- nrow(iris)
dat <- data.table::as.data.table(x = iris[sample(x = 1:n, size = n, replace = FALSE),])
dt.remove.variables(dt.name = "dat", the.variables = c("Category", "setosa_sl_below_5"),
return.as = "all")
```

---

## dt.sort *dt.sort*

---

### Description

This function sorts the rows of a data.frame or data.table based on selected columns. It is built as a light wrapper function of data.table's setorderv() function. Options also exist to return a data.table coding statement (result = "code") for educational purposes or both the result and the code together (result = "all"). For examples, please see the vignette.

## Usage

```
dt.sort(
  dt.name,
  sorting.variables,
  sort.increasing = TRUE,
  missing.variables = c("first", "last"),
  return.as = "result",
  envir = .GlobalEnv,
  ...
)
```

## Arguments

| | |
|---|---|
| dt.name | a character value specifying the name of a data.frame or data.table object to select data from. A variable called dat should be referred to with dt.name = "dat" when using the function. |
| sorting.variables | A vector specifying the variables that we want to sort by. For character vectors, only values that exist in the names of the data will be used. For numeric vectors, only the values of unique(floor(sorting.variables)) that are in 1:ncol() of your data will be used. Then these indices will be used to select column names from the data. Other values in sorting.variables that do not correspond to a defined column will be excluded from the calculation. The sorting proceeds in the order that sorting.variables is specified. |
| sort.increasing | A logical vector or numeric vector specifying whether the sorting should be increasing (TRUE or 1) or decreasing (FALSE or not 1) for each variable in sorting.variables. A vector such as c(TRUE, FALSE) would sort the first variable in increasing order and the second in decreasing order. If only a single value is provided (either TRUE or FALSE), then all of the.variables will be sorted in the specified ordering. |
| missing.variables | a character value of either "first" or "last" specifying where rows with missing values in the.variables should be included. Using "first" will place those rows at the beginning of the table, while "last" would place them in the end of the table. |
| return.as | a character value specifying what output should be returned. return.as = "result" provides the updated data. return.as = "code" provides a data.table coding statement. return.as = "all" provides a list object including both the resulting output and the code. |
| envir | a specification of the environment in which the data (referenced by dt.name) exists, with the global environment as the default value. |
| ... | additional arguments if required |

## Value

Depending on the value of return.as, the output will be a) a character value (return.as = 'code'), b) a coding output, typically a data.table (return.as = 'result'), or c) a list containing both the code and output (return.as = 'all')

## Examples

```
n <- nrow(iris)
dat <- data.table::as.data.table(x = iris[sample(x = 1:n, size = n, replace = FALSE),])
dt.sort(dt.name = "dat", sorting.variables = c("Species", "Sepal.Length"),
sort.increasing = TRUE, return.as = "all")
```

---

`reduce.vector.expression`

*Takes a numeric vector and produces a statement with a more compact representation. For instance, c(1,2,3,4) would become '1:4' and c(1:3, 4:6) could become '1:6'.*

---

## Description

Takes a numeric vector and produces a statement with a more compact representation. For instance, c(1,2,3,4) would become '1:4' and c(1:3, 4:6) could become '1:6'.

## Usage

```
reduce.vector.expression(x)
```

## Arguments

| | |
|---|---|
| x | a numeric vector |

## Value

None

# Index