

Package ‘datascan’

October 6, 2026

Title Scan Data for Quick Structural Summaries and Checks

Version 0.1.1

Description Scans data for checking columns that are constant, one-to-one, missing or all unique.
Users can also try to identify the columns that uniquely index the observational unit and whether some columns are nested or complete.

License MIT + file LICENSE

Encoding UTF-8

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Imports cli, dplyr, glue, magrittr, pillar, rlang, tibble, tidyr,
tidyselect

Depends R (>= 4.1.0)

LazyData true

URL <https://github.com/emitanaka/datascan>,
<https://emitanaka.org/datascan/>

BugReports <https://github.com/emitanaka/datascan/issues>

NeedsCompilation no

Author Emi Tanaka [aut, cre, cph] (ORCID:
<<https://orcid.org/0000-0002-1455-259X>>)

Maintainer Emi Tanaka <dr.emi.tanaka@gmail.com>

Repository CRAN

Date/Publication 2026-10-06 07:40:28 UTC

Contents

add_row_id	2
as_numerical	3
cols_all_unique	3

cols_bijective	4
cols_binary	5
cols_constant	5
cols_identify_all	6
cols_missing	7
cols_nested	8
concurrence_matrix	9
is_complete	10
is_nested	11
is_numerical	12
is_observational_unit	12
rows_missing	13
watch_tits	14
white_clover	15
Index	17

add_row_id	<i>Add a row ID column to a data frame</i>
------------	--

Description

This function adds a new column to a data frame containing a unique identifier for each row. The default column name is ".id", but this can be customized using the name parameter.

Usage

```
add_row_id(data, name = ".id")
```

Arguments

- data A data frame to which the row ID column will be added.
- name The name of the new row ID column. Default is ".id".

Value

The input data frame with an additional column containing row IDs.

Examples

```
df <- data.frame(x = c(1, 2, 3), y = c("a", "b", "c"))
add_row_id(df)
```

`as_numerical`*Coerce to numerical and identify non-numerical entries*

Description

This function attempts to coerce a variable to numerical and identifies any non-numerical entries. It returns the coerced numerical vector and prints a message indicating any non-numerical entries found.

Usage

```
as_numerical(x)
```

Arguments

`x` A vector to be coerced to numerical.

Details

Factors are coerced using their labels rather than their underlying integer codes.

Value

A numerical vector with non-numerical entries coerced to NA.

See Also

Other numerical: [is_numerical\(\)](#)

Examples

```
as_numerical(c("1", "2", "A", "4", "B"))
as_numerical(factor(c("10", "20", "A")))
```

`cols_all_unique`*Identify columns with all unique values*

Description

Find columns that have all unique values. If there are no identified columns, it will return NA.

Usage

```
cols_all_unique(data)
```

Arguments

data The data frame

Value

A character vector of column names with all unique values.

See Also

Other quality checks: [cols_bijective\(\)](#), [cols_constant\(\)](#), [cols_identify_all\(\)](#), [cols_missing\(\)](#), [cols_nested\(\)](#)

Examples

```
cols_all_unique(ChickWeight)
```

cols_bijective	<i>Identify bijective (one-to-one correspondence) columns</i>
----------------	---

Description

Any values that have all unique values will not be included in the output. If there are no identified columns, it will return an empty list.

Usage

```
cols_bijective(data)
```

Arguments

data The data frame

Value

A list of character vectors of bijective columns.

See Also

Other quality checks: [cols_all_unique\(\)](#), [cols_constant\(\)](#), [cols_identify_all\(\)](#), [cols_missing\(\)](#), [cols_nested\(\)](#)

Examples

```
cols_bijective(OrchardSprays)
```

cols_binary	<i>Identify columns with a specific number of unique values</i>
-------------	---

Description

This function identifies columns in a data frame that have a specific number of unique values. It can be used to find constant, binary, ternary, or any other n-ary columns. If there are no identified columns, it will return NA.

Usage

```
cols_binary(data, na.rm = FALSE)

cols_ternary(data, na.rm = FALSE)

cols_multinary(data, na.rm = FALSE, n = 1)
```

Arguments

data	The data frame
na.rm	Logical, whether to remove NA values before counting unique values.
n	The specific number of unique values to look for.

Value

A character vector of column names that have exactly n unique values.

Examples

```
cols_binary(mtcars)
```

cols_constant	<i>Identify constant columns</i>
---------------	----------------------------------

Description

Similar to `janitor::remove_constant`, except this function aims to identify the constant columns without removing them. If there are no constant columns, it will return NA.

Usage

```
cols_constant(data, na.rm = FALSE)

cols_unary(data, na.rm = FALSE)
```

Arguments

data	The data frame
na.rm	Should missing values be removed?

Value

A character vector of constant columns.

See Also

Other quality checks: [cols_all_unique\(\)](#), [cols_bijective\(\)](#), [cols_identify_all\(\)](#), [cols_missing\(\)](#), [cols_nested\(\)](#)

Examples

```
df <- data.frame(x1 = 1, x2 = c("A", "B", "C"), x3 = "a")
cols_constant(df)
```

cols_identify_all	<i>Identify all columns with potential issues</i>
-------------------	---

Description

This function identifies all columns with potential issues, including all unique values, bijective columns, constant columns, and columns with missing values above a certain cutoff. It returns a list of identified columns for each category and can print the results to the console.

Usage

```
cols_identify_all(data, cutoff = 1, na.rm = FALSE, print = TRUE)
```

Arguments

data	The data frame
cutoff	The minimum cutoff for the proportion of missing values.
na.rm	Should missing values be removed?
print	Print the output or not.

Value

A named list with one element per check.

See Also

Other quality checks: [cols_all_unique\(\)](#), [cols_bijective\(\)](#), [cols_constant\(\)](#), [cols_missing\(\)](#), [cols_nested\(\)](#)

Examples

```
cols_identify_all(airquality)
```

cols_missing	<i>Identify columns that have certain proportion of missing values</i>
--------------	--

Description

Similar to `janitor::remove_empty`, except this function aims to identify the columns without removing them. If there are no identified columns, it will return NA.

Usage

```
cols_missing(data, cutoff = 1)
```

Arguments

data	The data frame
cutoff	The minimum cutoff for the proportion of missing values.

Value

A character vector of column names with missing values with at least a certain proportion.

See Also

Other quality checks: [cols_all_unique\(\)](#), [cols_bijective\(\)](#), [cols_constant\(\)](#), [cols_identify_all\(\)](#), [cols_nested\(\)](#)

Examples

```
# find columns that have any missing values
cols_missing(airquality, 0)
```

`cols_nested`*Nestedness checks*

Description

This function checks for nestedness in a data frame. It identifies columns that are nested within other columns, meaning that the values in one column are subsets of the values in another column. The function returns a list of identified nested columns and can print the results to the console.

Usage

```
cols_nested(data, na.rm = FALSE, ignore = c("constant", "unique", "bijective"))
```

Arguments

<code>data</code>	The data frame to be checked for nestedness.
<code>na.rm</code>	Remove NA values when checking for nestedness.
<code>ignore</code>	A character vector of trivial cases of nesting to ignore, i.e. not count as nested. Any of "constant" (x or y has a single value), "unique" (x has all unique values) and "bijective" (x and y have a one-to-one correspondence). Use NULL to count all cases.

Details

By default, trivial cases of nesting (constant columns, columns with all unique values and pairs of bijective columns) are not reported as nested. Use `ignore` to change this (see [is_nested\(\)](#)).

Value

A list of identified nested columns. The first element of each list is the child variable, and the second element is the parent variable.

See Also

Other quality checks: [cols_all_unique\(\)](#), [cols_bijective\(\)](#), [cols_constant\(\)](#), [cols_identify_all\(\)](#), [cols_missing\(\)](#)

Examples

```
cols_nested(CO2)
```

concurrency_matrix	<i>Get the concurrency matrix or table</i>
--------------------	--

Description

Find the number of common levels of first variable across the levels of the second variable. The diagonal shows the total number of levels at the corresponding level. The off-diagonal shows the number of common levels between the two levels. The matrix is symmetric.

Usage

```
concurrency_matrix(data, x, group, na.rm = FALSE)
```

```
concurrency_table(data, x, group, na.rm = FALSE)
```

Arguments

data	A data frame containing the MET data
x	Categorical variable to be used for the concurrency matrix.
group	Categorical grouping variable.
na.rm	Remove NA values when checking for concurrency.

Value

`concurrency_matrix()` returns a symmetric numeric matrix of class `concurrency_mat` with one row and one column per level of `group`. The diagonal entries give the number of distinct levels of `x` observed at each level of `group`, and the off-diagonal entries give the number of levels of `x` shared by the two corresponding levels of `group`. The names of `x` and `group` are stored in the `.vars` attribute.

`concurrency_table()` returns the same information in long format as a tibble of class `concurrency_tbl`, with one row per pair of group levels and the columns:

```
<group>_1, <group>_2 Factors giving the pair of group levels.
concurrency Number of levels of x shared by the pair.
prop_in_1 Proportion of the levels of x in <group>_1 that are also in <group>_2.
prop_in_2 Proportion of the levels of x in <group>_2 that are also in <group>_1.
```

Examples

```
df <- expand.grid(gen = paste0("G", 1:20), env = paste0("E", 1:15))
df <- df[sample(nrow(df), 100), ]
(mat <- concurrency_matrix(df, gen, env))
extract(mat, "E1", "E2")
extract(mat, "E1", c("E2", "E3"))
extract(mat, c("E5", "E6", "E7"), c("E2", "E3"))
# proportion of levels in the first variable
```

```
# at each level of the second variable
proportions(mat, 1)
```

is_complete

Is variable complete within another variable?

Description

This function checks if a variable is complete within another variable. If all levels of the first variable appear in all levels of the second variable, it returns TRUE; otherwise, it returns FALSE. The function can also handle NA values based on the `na.rm` parameter.

Usage

```
is_complete(x, y, na.rm = TRUE)
```

Arguments

<code>x, y</code>	Vectors of the same size where <code>x</code> (child variable) is checked for completeness within <code>y</code> (parent variable)
<code>na.rm</code>	Remove NA values when checking for completeness.

Value

TRUE if the values in vector `x` are complete

Examples

```
df <- data.frame(
  site = c("A", "A", "B", "B"),
  plot1 = c("A1", "A2", "B1", "B2"),
  plot2 = c("A1", "A2", "A1", "A2")
)
is_complete(df$plot1, df$site) # FALSE
is_complete(df$plot2, df$site) # TRUE
```

is_nested

*Is variable nested within another variable?***Description**

This function checks if a variable is nested within another variable. If the levels in the first variable appear only within the levels of the second variable, it returns TRUE; otherwise, it returns FALSE. The function can also handle NA values based on the `na.rm` parameter.

Usage

```
is_nested(x, y, na.rm = TRUE, ignore = c("constant", "unique", "bijective"))
```

Arguments

<code>x, y</code>	Vectors of the same size where <code>x</code> (child variable) is nested in <code>y</code> (parent variable)
<code>na.rm</code>	Remove NA values when checking for nestedness.
<code>ignore</code>	A character vector of trivial cases of nesting to ignore, i.e. not count as nested. Any of "constant" (<code>x</code> or <code>y</code> has a single value), "unique" (<code>x</code> has all unique values) and "bijective" (<code>x</code> and <code>y</code> have a one-to-one correspondence). Use NULL to count all cases.

Details

By default, trivial cases of nesting are not counted and it returns FALSE when `x` or `y` is constant (has a single value), `x` has all unique values, or `x` and `y` are bijective (have a one-to-one correspondence). Use `ignore` to choose which of these cases are not counted.

Value

TRUE if the values in vector `x` are nested within vector `y`, FALSE otherwise. It is also FALSE for any trivial case listed in `ignore`.

Examples

```
df <- data.frame(
  site = rep(c("A", "B"), each = 4),
  plot1 = rep(c("A1", "A2", "B1", "B2"), each = 2),
  plot2 = rep(c("A1", "A2", "A1", "A2"), each = 2)
)
is_nested(df$plot1, df$site) # TRUE
is_nested(df$plot2, df$site) # FALSE since A1 and A2 appear in both sites
df$id <- seq_len(nrow(df))
is_nested(df$id, df$site) # FALSE since id has all unique values
is_nested(df$id, df$site, ignore = c("constant", "bijective")) # TRUE
```

is_numerical	<i>Is the variable numerical?</i>
--------------	-----------------------------------

Description

This function checks if a variable is numerical. It returns TRUE if the variable is numerical and FALSE otherwise. If non-numerical entries are found, it prints a message indicating the non-numerical entries.

Usage

```
is_numerical(x)
```

Arguments

x A vector to be checked for numericality.

Value

TRUE if the variable is numerical, FALSE otherwise.

See Also

Other numerical: [as_numerical\(\)](#)

Examples

```
is_numerical(c("1", "2", "3"))
is_numerical(c("1", "2", "3", "A"))
```

is_observational_unit	<i>Is it the observational unit?</i>
-----------------------	--------------------------------------

Description

An observational unit is the entity that is being measured in a study. For example, if blood pressure of a person is measured, then the person is the observational unit. If blood pressure is measured multiple times for a person, then the observational unit is the person and the time of measurement.

Usage

```
is_observational_unit(.data, ..., .check_redundancy = TRUE)
```

```
is_obs_unit(.data, ..., .check_redundancy = TRUE)
```

Arguments

`.data` The data frame

`...` The columns in the data frame that potentially index the observational unit.

`.check_redundancy` Logical. If TRUE, checks if any of the columns are redundant.

Details

Supply the columns that potentially index the observational unit. The column cannot uniquely identify the row. If the number of rows in the data frame is equal to the number of distinct rows in the selected columns, then it returns TRUE, otherwise FALSE. If TRUE, it checks if any of the columns are redundant.

Value

A single logical value. TRUE if the selected columns uniquely identify each row of `.data` (and, when `.check_redundancy = TRUE`, no selected column is redundant), and FALSE otherwise. In an interactive session, a message with the result is also printed.

Examples

```
is_observational_unit(ChickWeight, Time, Chick)
is_observational_unit(ChickWeight, Time, Chick, Diet)
```

rows_missing	<i>Identify rows that have certain proportion of missing values</i>
--------------	---

Description

Similar to `janitor::remove_empty`, except this function aims to identify the columns without removing them. If there are no identified columns, it will return NA.

Usage

```
rows_missing(data, cutoff = 1)
```

Arguments

`data` The data frame

`cutoff` The minimum cutoff for the proportion of missing values.

Value

An integer vector of rows with missing values with at least a certain proportion.

Examples

```
rows_missing(airquality, 0.1)
```

watch_tits	<i>Coordination of care by breeders and helpers in the cooperatively breeding long-tailed tit</i>
------------	---

Description

- **id**: Unique identifier for each provisioning watch
- **feeds**: Number of feeds by each individual per watch
- **data_type**: Whether the data was taken directly from field observation (observed) or from null model randomization (expected)
- **alt_feeds**: Number of alternated visits
- **percent_alt**: Percentage of each carer's visits which were alternated
- **sync_feeds**: Number of synchronized visits (2-minute interval)
- **percent_sync2mins**: Percentage of each carer's visits which were synchronized (2-minute interval)
- **carer_status**: Factor designating whether a carer was a breeding female (female), breeding male (male) or non-breeding helper (helper)
- **carer_id**: The unique identifier for each carer
- **total_feed_rate**: The number of feeds performed by all carers per hour
- **brood_size**: The number of chicks in the brood
- **carer_number**: The number of carers observed provisioning during a given watch
- **watch_duration_mins**: The time, in minutes, between the first and last recorded feed during a given watch
- **watch_time**: Number of hours since the start of the day 00:00 the watch was started
- **brood_age**: The number of days, since recorded hatching, that a watch was performed
- **julian_hatch_date**: The number of days since March 1 each year that the eggs hatched
- **a_max**: The total percentage of visits during a given watch which could theoretically be alternated (or synchronized). Formula as follows: $100 - ((\text{Number of feeds by highest feed rate carer} - \text{Number of feeds by all remaining carers}) / \text{Total number of feeds}) * 100$
- **year**: Unique identifier for each year the watch was performed during
- **nest**: Unique identifier for each nest
- **individual_feed_rate**: The number of feeds performed by each carer per hour
- **row_ref**: Row reference 1-7950

Usage

```
watch_tits
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 7950 rows and 21 columns.

Source

Chay et al. (2022) "Coordination of care by breeders and helpers in the cooperatively breeding long-tailed tit, *Aegithalos caudatus*". Data available at Dryad: <https://datadryad.org/dataset/doi:10.5061/dryad.mkkwh712k>

white_clover	<i>A field experiment assessing the roles of drought, herbivory, and local climate for white clovers</i>
--------------	--

Description

This dataset contains data from a field experiment conducted by Albano et al. (2026) to assess the roles of drought, herbivory, and local climate on white clovers (*Trifolium repens*). The experiment was designed to investigate how these factors influence the growth, survival, and reproductive success of white clovers in different environmental conditions.

Usage

```
white_clover
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 768 rows and 27 columns.

Details

- Garden: Location of the experimental site (Ontario or Louisiana).
- WholePlotID: Identifier of the whole plot, each containing 8 plants (numbered 1 to 48).
- SplitPlotID: Identifier of the split plot within each whole plot, each containing 4 plants (numbered 1 or 2).
- PlantID: Identifier of the plant within each split plot (numbered 1 to 4).
- Cyanotype: One of four phenotypes (AcLi, Acli, acLi or acli) based on the presence or absence of a dominant allele at each of the two loci (Ac/ac and Li/li) underlying hydrogen cyanide (HCN) production.
- Cyanogenesis: Whether the plant can produce HCN: Cyanogenic (AcLi) or Acyanogenic (Acli, acLi or acli).
- Ac_ac: Whether the plant has at least one dominant allele at the Ac/ac locus (Ac = yes, ac = no).
- Li_li: Whether the plant has at least one dominant allele at the Li/li locus (Li = yes, li = no).
- Precipitation: Precipitation reduction treatment (Control or Reduced).
- Herbivores: Herbivore reduction treatment (Control or Reduced).

- **Survived21:** Whether the plant survived the first growing season (2021); 1 = yes, 0 = no.
- **Flowered21:** Whether the plant flowered in the first growing season (2021); 1 = yes, 0 = no.
- **Seeded21:** Whether the plant produced seeds in the first growing season (2021); 1 = yes, 0 = no.
- **FlowerHeadNumber21:** Total number of flower heads produced by the plant in the first growing season (2021).
- **SeedSetMass21:** Total seed set mass (in grams) from the flower heads collected from the plant in the first growing season (2021).
- **MaxArea21:** Maximum lateral area (in cm^2) taken up by the plant across the monthly measurements of the first growing season (2021).
- **GrowthRate21:** Growth rate of the plant during the first growing season (2021), calculated as $\ln(\text{maximum lateral area}) - \ln(\text{initial lateral area})$, divided by the number of days between these measurements.
- **Herbivory21:** Percentage of leaf area consumed by herbivores, averaged across 5 trifoliate leaves per plant, in July of the first growing season (2021).
- **SurvivedWinter:** Whether the plant survived the winter between the 2021 and 2022 growing seasons; 1 = yes, 0 = no. This is NA for all plants in Louisiana and for plants in Ontario that died during the 2021 growing season.
- **Survived22:** Whether the plant survived the second growing season (2022); 1 = yes, 0 = no.
- **Flowered22:** Whether the plant flowered in the second growing season (2022); 1 = yes, 0 = no.
- **Seeded22:** Whether the plant produced seeds in the second growing season (2022); 1 = yes, 0 = no.
- **FlowerHeadNumber22:** Total number of flower heads produced by the plant in the second growing season (2022).
- **SeedSetMass22:** Total seed set mass (in grams) from the flower heads collected from the plant in the second growing season (2022).
- **MaxArea22:** Maximum lateral area (in cm^2) taken up by the plant across the monthly measurements of the second growing season (2022).
- **GrowthRate22:** Growth rate of the plant during the second growing season (2022), calculated in the same way as GrowthRate21.
- **Herbivory22:** Percentage of leaf area consumed by herbivores, averaged across 5 trifoliate leaves per plant, in the second growing season (2022).

Missing values (NA) in the 2021 and 2022 measurements represent plants for which data could not be collected, or, for 2022, plants that had already died.

Source

Albano et al. (2026) "Data from: A field experiment assessing the roles of drought, herbivory, and local climate on cyanogenesis cline formation and local adaptation in *Trifolium repens*". Data available at Dryad: <https://datadryad.org/dataset/doi%3A10.5061/dryad.4xgxd25n1>

Index

- * **datasets**
 - watch_tits, [14](#)
 - white_clover, [15](#)
- * **numerical**
 - as_numerical, [3](#)
 - is_numerical, [12](#)
- * **quality checks**
 - cols_all_unique, [3](#)
 - cols_bijective, [4](#)
 - cols_constant, [5](#)
 - cols_identify_all, [6](#)
 - cols_missing, [7](#)
 - cols_nested, [8](#)
- * **quality-checks**
 - cols_binary, [5](#)

add_row_id, [2](#)
as_numerical, [3](#)
as_numerical(), [12](#)

cols_all_unique, [3](#)
cols_all_unique(), [4](#), [6–8](#)
cols_bijective, [4](#)
cols_bijective(), [4](#), [6–8](#)
cols_binary, [5](#)
cols_constant, [5](#)
cols_constant(), [4](#), [6–8](#)
cols_identify_all, [6](#)
cols_identify_all(), [4](#), [6–8](#)
cols_missing, [7](#)
cols_missing(), [4](#), [6](#), [8](#)
cols_multinary(cols_binary), [5](#)
cols_nested, [8](#)
cols_nested(), [4](#), [6](#), [7](#)
cols_ternary(cols_binary), [5](#)
cols_unary(cols_constant), [5](#)
concurrence_matrix, [9](#)
concurrence_table(concurrence_matrix),
[9](#)

is_complete, [10](#)
is_nested, [11](#)
is_nested(), [8](#)
is_numerical, [12](#)
is_numerical(), [3](#)
is_obs_unit(is_observational_unit), [12](#)
is_observational_unit, [12](#)

rows_missing, [13](#)

watch_tits, [14](#)
white_clover, [15](#)