Package 'seplyr'

September 25, 2018

Type Package

Title Improved Standard Evaluation Interfaces for Common Data Manipulation Tasks

Version 0.8.2

Date 2018-09-25

Maintainer John Mount <jmount@win-vector.com>

URL https://github.com/WinVector/seplyr/, https://winvector.github.io/seplyr/

BugReports https://github.com/WinVector/seplyr/issues

Description The 'seplyr' (standard evaluation plying) package supplies improved standard evaluation adapter methods for important common 'dplyr' data manipulation tasks. In addition the 'seplyr' package supplies several new ``key operations bound together'' methods. These include 'group_summarize()' (which combines grouping, arranging and calculation in an atomic unit), 'add_group_summaries()' (which joins grouped summaries into a 'data.frame' in a well documented manner), 'add_group_indices()' (which adds per-group identifiers to a 'data.frame' without depending on row-order), 'partition_mutate_qt()' (which optimizes mutate sequences), and 'if_else_device()' (which simulates per-row if-else blocks in expression sequences).

License GPL-3

Encoding UTF-8

Depends R (>= 3.4.0), wrapr (>= 1.6.2)

Imports dplyr (>= 0.7.0), rlang (>= 0.2.0), tidyr

LazyData true

RoxygenNote 6.1.0

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr

ByteCompile true

NeedsCompilation no

Author John Mount [aut, cre],
Win-Vector LLC [cph]

Repository CRAN

Date/Publication 2018-09-25 12:50:02 UTC

R topics documented:

add_count_se *tally/count standard interface.*

Description

Add a new column named "n" with (optionally per-group) sums/counts.

Usage

```
add_count_se(x, groupingVars = NULL, wt = NULL, sort = FALSE)
```

Arguments

x	data.frame to tally/count
groupingVars	character vector of column names to group by.
wt	character optional column name containing row-weights (passed to count/tally)
sort	logical if TRUE sort result in descending order

Details

Note: dplyr::count, dplyr::add_count, dplyr::tally, and dplyr::add_tally are not S3 methods, so it may not be practical to re-dispatch seplyr calls to these dplyr implementations.

Value

.data with added column n, containing counts.

See Also

add_count

Examples

```
datasets::iris %.>% count_se(., wt = "Sepal.Width", groupingVars= c('Species'))
```

add_group_indices *Group a data frame and add per-group indices as a column.*

Description

Group a data frame and add per-group indices as a column.

Usage

```
add_group_indices(.data, groupingVars, indexColumn)
```

Arguments

.data	data.frame
groupingVars	character vector of column names to group by.
indexColumn	character name of column to add indices to.

Value

.data with group identifying column added.

Examples

```
add_group_indices(datasets::mtcars, c("cyl", "gear"), 'groupID')
```

add_group_sub_indices *Group a data frame and add in-group indices as a column.*

Description

Group a data frame and add in-group indices as a column.

Usage

```
add_group_sub_indices(.data, ..., groupingVars, orderColumn,
  arrangeTerms = NULL, env = parent.frame())
```

Arguments

.data	data.frame
...	force later arguments to bind by name.
groupingVars	character vector of column names to group by.
orderColumn	character name of column to add in-group order marks to.
arrangeTerms	character vector of column expressions to arrange by.
env	environment to work in.

Value

.data with in group order indices added (no ties).

Examples

```
groupingVars = c("cyl", "gear")

datasets::mtcars %.>%
  # dplyr doesn't currently export tibble::rownames_to_column()
  mutate_se(., "CarName" := "rownames(.)" ) %.>%
  select_se(., c('CarName', 'cyl', 'gear', 'hp', 'wt')) %.>%
  add_group_indices(., groupingVars = groupingVars,
                    indexColumn = 'groupID') %.>%
  add_group_sub_indices(., groupingVars = groupingVars,
                        arrangeTerms = c('desc(hp)', 'wt'),
                        orderColumn = 'orderInGroup') %.>%
  arrange_se(., c('groupID', 'orderInGroup'))
```

---

add_group_summaries *Simulate the group_by/mutate pattern with an explicit summarize and join.*

Description

Group a data frame by the groupingVars and compute user summaries on this data frame (user summaries specified in ...), then join these new columns back into the original data and return to the user. Author: John Mount, Win-Vector LLC.

Usage

```
add_group_summaries(d, groupingVars, ..., arrangeTerms = NULL)
```

Arguments

d	data.frame
groupingVars	character vector of column names to group by.
...	list of dplyr::mutate() expressions.
arrangeTerms	character optional vector of column expressions to arrange by.

Value

d with grouped summaries added as extra columns

Examples

```
add_group_summaries(datasets::mtcars,
                    c("cyl", "gear"),
                    group_mean_mpg = mean(mpg),
                    group_mean_disp = mean(disp)) %.>%
  head(.)
```

`add_rank_indices` *Arrange a data frame and rank indexes.*

Description

Arrange a data frame and rank indexes.

Usage

```
add_rank_indices(.data, ..., arrangeTerms = NULL, orderColumn)
```

Arguments

`.data`	data.frame
`...`	force later arguments to bind by name.
`arrangeTerms`	character vector of column expressions to arrange by.
`orderColumn`	character name of column to add in-group order marks to.

Value

.data with order indices added (no ties).

Examples

```
datasets::mtcars %.>%
  # tibble::rownames_to_column() not currently re-exported by dplyr
  mutate_se(., "CarName" := "rownames(.)" ) %.>%
  select_se(., c('CarName', 'hp', 'wt')) %.>%
  add_rank_indices(., arrangeTerms = c('desc(hp)', 'wt'),
                   orderColumn = 'rankID') %.>%
  arrange_se(., 'rankID')
```

add_tally_se *tally/count standard interface.*

Description

Add a new column named "n" with (optionally per-group) sums/counts.

Usage

```
add_tally_se(x, wt = NULL, sort = FALSE)
```

Arguments

x	data.frame to tally/count
wt	character optional column name containing row-weights (passed to count/tally)
sort	logical if TRUE sort result in descending order

Details

Note: `dplyr::count`, `dplyr::add_count`, `dplyr::tally`, and `dplyr::add_tally` are not `S3` methods, so it may not be practical to re-dispatch `seplyr` calls to these `dplyr` implementations.

Value

.data with added column n, containing counts.

See Also

`add_tally`

Examples

```
datasets::iris %.>% add_tally_se(.)
```

arrange_se *Arrange standard interface.*

Description

Arrange a data frame by the possibly the `group_vars()` (optional, but defaults to off) and `arrangeTerms`. Accepts arbitrary text as arrangeTerms to allow forms such as "desc(gear)". Intent is to arrange only by sets of variables with desc() notations reversals, not by arbitrary expressions over variables. To help enforce this parsing is performed in an empty environment (so expressions such as "gear + carb" deliberately error-out).

Usage

```
arrange_se(.data, arrangeTerms, ..., .by_group = FALSE, strict = TRUE)
```

Arguments

`.data`	data.frame
`arrangeTerms`	character vector of column expressions to arrange by.
`...`	not used, force later arguments to bind by name.
`.by_group`	logical, should data be sorted by grouping variables (if present).
`strict`	logical if TRUE accept only name and desc(name) terms.

Value

.data arrnaged by arrangeTerms

See Also

arrange, arrange_at

Examples

```
datasets::mtcars %.>%
  arrange_se(., c("cyl", "desc(gear)")) %.>%
  head(.)
# equivilent to dplyr/magrittr pipeline
# arrange(datasets::mtcars, cyl, desc(gear)) %>% head()

# Note: arranging in the presence of groups is subtle.
# As grouping is an annotation, not an ordering (and ordering is
# unfortunately not an annotation).

d <- data.frame(x = 1:6,
                sin_x = sin(1:6),
                grp = rep(c("a", "b"), 3),
                stringsAsFactors = FALSE)
```

```
# arranged by sin_x and not by grp
d %.>%
  group_by_se(., "grp") %.>%
  arrange_se(., "sin_x")

# arranged by sin_x and not by grp
d %.>%
  arrange_se(., "sin_x") %.>%
  group_by_se(., "grp")

# arranged by sin_x and not by grp
d %.>%
  group_by_se(., "grp") %.>%
  arrange_se(., "sin_x", .by_group = TRUE)

# arranged by sin_x and not by grp
d %.>%
  arrange_se(., "sin_x", .by_group = TRUE) %.>%
  group_by_se(., "grp")
```

`complete_se` *complete by standard interface*

Description

Complete a data frame with missing combinations of data. Turns implicit missing values into explicit missing values.

Usage

```
complete_se(data, col_terms, fill = list(), env = parent.frame())
```

Arguments

`data`	A data frame or tbl.
`col_terms`	A character vector of column names or expressions to complete by.
`fill`	A list that for each variable supplies a single value to use instead of NA for missing combinations.
`env`	The environment as an argument (in case the function is called from another function).

Details

This is a standard evaluation interface for `tidyr::complete()`. The purpose of the function is to be able to use a vector of characters (column names) as the argument for expanding the data frame.

Value

The data frame with implicit missing values identified.

Examples

```
# data frame used to illustrate tidyr::complete()
library(dplyr, warn.conflicts = FALSE)
library(tidyr) # for nesting()
df <- tibble(
group = c(1:2, 1),
item_id = c(1:2, 2),
item_name = c("a", "b", "b"),
value1 = 1:3,
value2 = 4:6)

# columns to complete by
col_terms <- c("group", "item_id", "item_name")
df %.>% complete_se(., col_terms)
df %.>% complete_se(., col_terms, fill = list(value1 = 0))

# with nesting
col_terms <- c("group", "nesting(item_id, item_name)")
df %.>% complete_se(., col_terms)
df %.>% complete_se(., col_terms, fill = list(value1 = 0))
df %.>% complete_se(., col_terms, fill = list(value1 = 0, value2 = 0))
```

---

count_se *tally/count standard interface.*

Description

Add a new column named "n" with (optionally per-group) sums/counts.

Usage

```
count_se(x, groupingVars = NULL, wt = NULL, sort = FALSE)
```

Arguments

x	data.frame to tally/count
groupingVars	character vector of column names to group by.
wt	character optional column name containing row-weights (passed to count/tally)
sort	logical if TRUE sort result in descending order

Details

Note: `dplyr::count`, `dplyr::add_count`, `dplyr::tally`, and `dplyr::add_tally` are not `S3` methods, so it may not be practical to re-dispatch `seplyr` calls to these `dplyr` implementations.

Value

.data with added column n, containing counts.

See Also

count

Examples

```
datasets::mtcars %.>% count_se(., groupingVars= c('cyl', 'gear'))
```

---

deselect *deselect standard interface.*

Description

deselect columns. To keep columns please see select_se.

Usage

```
deselect(.data, colNames)
```

Arguments

.data	data.frame
colNames	character vector of columns to remove

Value

.data without deselected columns

See Also

select_se, select, select_at

Examples

```
datasets::mtcars %.>%
   deselect(., c("cyl", "gear")) %.>%
   head(.)
# essentially dplyr::select( datasets::mtcars, -cyl, -gear)
```

distinct_se *Standard interface for distinct.*

Description

Group a data frame and add per-group indices as a column.

Usage

```
distinct_se(.data, groupingVars, .keep_all = FALSE)
```

Arguments

.data	data.frame
groupingVars	character vector of column names to group by.
.keep_all	logical, passed to dplyr::distinct.

Value

.data passed through distinct with groupingVars args.

See Also

distinct

Examples

```
datasets::mtcars %.>% distinct_se(., c("cyl", "gear"))
```

factor_mutate *Re-write a* dplyr::mutate() *into safe blocks.*

Description

Note: not for use with rlang expressions (guesses variable names by text inspection). See also: https://winvector.github.io/rquery/articles/AssigmentPartitioner.html.

Usage

```
factor_mutate(..., factor_mutate_warn_msg = TRUE)
```

Arguments

`...`	mutate terms
`factor_mutate_warn_msg`	logical if TRUE issue a warning message on non-trivial mutates.

Value

partitioned dplyr::mutate() source text

Examples

```
cat(factor_mutate(
 choice_a = rand_a >= 0.5,
   a_1 = ifelse(choice_a, 'T', 'C'),
   a_2 = ifelse(choice_a, 'C', 'T'),
  choice_b = rand_b >= 0.5,
   b_1 = ifelse(choice_b, 'T', 'C'),
   b_2 = ifelse(choice_b, 'C', 'T'),
  choice_c = rand_c >= 0.5,
   c_1 = ifelse(choice_c, 'T', 'C'),
   c_2 = ifelse(choice_c, 'C', 'T'),
  choice_d = rand_d >= 0.5,
   d_1 = ifelse(choice_d, 'T', 'C'),
   d_2 = ifelse(choice_d, 'C', 'T'),
  choice_e = rand_e >= 0.5,
   e_1 = ifelse(choice_e, 'T', 'C'),
   e_2 = ifelse(choice_e, 'C', 'T'),
 factor_mutate_warn_msg = FALSE ))

cat(factor_mutate(
 choice = rand_a >= 0.5,
   a_1 = ifelse(choice, 'T', 'C'),
   a_2 = ifelse(choice, 'C', 'T'),
  choice = rand_b >= 0.5,
   b_1 = ifelse(choice, 'T', 'C'),
   b_2 = ifelse(choice, 'C', 'T'),
  choice = rand_c >= 0.5,
   c_1 = ifelse(choice, 'T', 'C'),
   c_2 = ifelse(choice, 'C', 'T'),
  choice = rand_d >= 0.5,
   d_1 = ifelse(choice, 'T', 'C'),
   d_2 = ifelse(choice, 'C', 'T'),
  choice = rand_e >= 0.5,
   e_1 = ifelse(choice, 'T', 'C'),
   e_2 = ifelse(choice, 'C', 'T'),
 factor_mutate_warn_msg = FALSE))
```

filter_nse *Filter non-standard interface.*

Description

Filter a data frame by the filter terms in ....

Usage

```
filter_nse(.data, ..., filter_nse_env = parent.frame())
```

Arguments

.data	data.frame
...	stringified expressions to filter by.
filter_nse_env	environment to work in.

Value

.data filtered by columns named in filterTerms

See Also

filter_se, filter, filter_at

Examples

```
upperBound <- 3.5

datasets::iris %.>%
  filter_nse(., Sepal.Length >= 2 * Sepal.Width,
                 Petal.Length <= upperBound)
```

filter_se *filter standard interface.*

Description

Filter a data frame by the filterTerms. Accepts arbitrary text as filterTerms to allow forms such as "Sepal.Length >= 2 * Sepal.Width".

Usage

```
filter_se(.data, filterTerms, env = parent.frame())
```

Arguments

.data	data.frame
filterTerms	character vector or list of column expressions to filter by.
env	environment to work in.

Value

.data filtered by columns named in filterTerms

See Also

filter, filter_at

Examples

```
upperBound <- 3.5

datasets::iris %.>%
  filter_se(., qe(Sepal.Length >= 2 * Sepal.Width,
                  Petal.Length <= upperBound))
```

---

gather_se *Distribute columns into blocks of rows.*

Description

A standard (value-oriented) interface for gather. Take values from the columns named in the columns argument and move them into blocks of rows, placing values in the new column specified by value and indicating which column each value came from in the new column specified by key.

Usage

```
gather_se(data, ..., key = "key", value = "value", columns = NULL,
  na.rm = FALSE, convert = FALSE, factor_key = FALSE,
  use_one_of = TRUE)
```

Arguments

data	data.frame to take values from.
...	not used, force later arguments to bind by name.
key	character, name for new column to record which columns values were taken from.
value	character, name for new column to record values.

columns	character, names of columns to take values from.
na.rm	passed to gather.
convert	passed to gather.
factor_key	passed to gather.
use_one_of	logical, if TRUE use dplyr::one_of() instead of rlang:::'!!¡.

Value

converted data.

See Also

gather, spread_se

Examples

```
d <- wrapr::build_frame(
    'id', 'measurement1', 'measurement2' |
    1   , 'a'           , 10             |
    2   , 'b'           , 20             )
gather_se(d,
  key = "value_came_from_column",
  value = "value_was",
  columns = c("measurement1", "measurement2"))
```

---

group_by_se *group_by standard interface.*

Description

Group a data frame by the groupingVars. group_by_se intentionally groups only by sets of variables, not by expressions over variables.

Usage

```
group_by_se(.data, groupingVars, add = FALSE)
```

Arguments

.data	data.frame
groupingVars	character vector of column names to group by.
add	logical, passed to group_by

Value

.data grouped by columns named in groupingVars

See Also

group_by, group_by_at

Examples

```
datasets::mtcars %.>%
  group_by_se(., c("cyl", "gear")) %.>%
  head(.)
```

---

group_indices_se *group_indices standard interface.*

Description

Group a data frame by the groupingVars and add group labels.

Usage

```
group_indices_se(.data, groupingVars, add = FALSE)
```

Arguments

.data	data.frame
groupingVars	character vector of column names to group by.
add	logical, passed to group_by

Value

per-row group index assignments

See Also

group_indices

Examples

```
group_indices_se(datasets::mtcars, c("cyl", "gear"))
```

`group_summarize` *group_by and summarize as an atomic action.*

Description

Group a data frame by the groupingVars and compute user summaries on this data frame (user summaries specified in ...). Enforces the good dplyr pipeline design principle of keeping group_by and summarize close together. Author: John Mount, Win-Vector LLC.

Usage

```
group_summarize(d, groupingVars, ..., arrangeTerms = NULL,
  env = parent.frame())

group_summarise(d, groupingVars, ..., arrangeTerms = NULL,
  env = parent.frame())
```

Arguments

`d`	data.frame
`groupingVars`	character vector of column names to group by.
`...`	list of dplyr::mutate() expressions.
`arrangeTerms`	character optional vector of column expressions to arrange by.
`env`	environment to work in.

Value

d summarized by groups

Examples

```
group_summarize(datasets::mtcars,
                    c("cyl", "gear"),
                    group_mean_mpg = mean(mpg),
                    group_mean_disp = mean(disp)) %.>%
  head(.)
```

`if_else_device` *Simulate a per-row block-*`if(){}else{}`.

Description

This device uses expression-`ifelse(,,)` to simulate the more powerful per-row block-`if(){}else{}`. The difference is expression-`ifelse(,,)` can choose per-row what value to express, whereas block-`if(){}else{}` can choose per-row where to assign multiple values. By simulation we mean: a sequence of quoted mutate expressions are emitted that implement the transform (versus a using a custom `dplyr` pipe stage or function). These expressions can then be optimized into a minimal number of no-dependency blocks by `partition_mutate_se` for efficient execution. The idea is the user can write legible code in this notation, and the translation turns it into safe and efficient code suitable for execution either on `data.frames` or at a big data scale using `RPostgreSQL` or `sparklyr`.

Usage

```
if_else_device(testexpr, thenexprs = NULL, elseexprs = NULL)
```

Arguments

`testexpr`	character containing the test expression.
`thenexprs`	named character then assignments (altering columns, not creating).
`elseexprs`	named character else assignments (altering columns, not creating).

Details

Note: `ifebtest_*` is a reserved column name for this procedure.

Examples

```
# Example: clear one of a or b in any row where both are set.
d <- data.frame(a = c(0, 0, 1, 1, 1, 1, 1, 1, 1, 1),
                b = c(0, 1, 0, 1, 1, 1, 1, 1, 1, 1),
                edited = FALSE)

program <- if_else_device(
  testexpr = '(a+b)>1',
  thenexprs = c(
    if_else_device(
      testexpr = 'runif(n()) >= 0.5',
      thenexprs = 'a' := '0',
      elseexprs = 'b' := '0'),
    'edited' := 'TRUE'))
print(program)

plan <- partition_mutate_se(program)
```

```
print(plan)

res <- d %.>%
  mutate_seb(., plan) %.>%
  select_se(., grepdf('^ifebtest_.*', ., invert=TRUE))
print(res)

## Note: with wrapr version 1.0.2 or greater
## you can write this without quotes code as:
# program <- if_else_device(
#  testexpr = qe((a+b)>1),
#  thenexprs = c(
#    if_else_device(
#      testexpr = qe(runif(n()) >= 0.5),
#      thenexprs = qae(a := 0),
#      elseexprs = qae(b := 0)),
#    qae(edited := TRUE)))
```

`mutate_nse` *mutate non-standard evaluation interface.*

Description

Mutate a data frame by the mutate terms from ....

Usage

```
mutate_nse(.data, ..., mutate_nse_split_terms = TRUE,
  mutate_nse_env = parent.frame(), mutate_nse_warn = TRUE,
  mutate_nse_printPlan = FALSE)
```

Arguments

`.data`	data.frame
`...`	expressions to mutate by.
`mutate_nse_split_terms`	logical, if TRUE into separate mutates (if FALSE instead, pass all at once to dplyr).
`mutate_nse_env`	environment to work in.
`mutate_nse_warn`	logical, if TRUE warn about name re-use.
`mutate_nse_printPlan`	logical, if TRUE print the expression plan

Details

Note: this method as the default setting `mutate_nse_split_terms = TRUE`, which is safer (avoiding certain known `dplyr`/`dblyr` issues) (please see the side-notes of http://winvector.github.io/FluidData/partition_mutate.html for some references).

Value

.data with altered columns.

See Also

`mutate_se`, `mutate`, `mutate_at`, `:=`

Examples

```
limit <- 3.5

datasets::iris %.>%
  mutate_nse(., Sepal_Long := Sepal.Length >= 2 * Sepal.Width,
                Petal_Short := Petal.Length <= limit) %.>%
  head(.)

# generates a warning
data.frame(x = 1, y = 2) %.>%
   mutate_nse(., x = y, y = x)
```

---

mutate_se *mutate standard evaluation interface.*

Description

Mutate a data frame by the mutateTerms. Accepts arbitrary text as mutateTerms to allow forms such as "Sepal.Length >= 2 * Sepal.Width". Terms are vectors or lists of the form "lhs := rhs". Semantics are: terms are evaluated left to right if splitTerms==TRUE (the default).

Usage

```
mutate_se(.data, mutateTerms, ..., splitTerms = TRUE, warn = TRUE,
  env = parent.frame(), printPlan = FALSE)
```

Arguments

.data	data.frame
mutateTerms	character vector of column expressions to mutate by.
...	force later terms to be bound by name
splitTerms	logical, if TRUE into separate mutates (if FALSE instead, pass all at once to dplyr).
warn	logical, if TRUE warn about name re-use.
env	environment to work in.
printPlan	logical, if TRUE print the expression plan.

Details

Note: this method as the default setting `splitTerms = TRUE`, which is safer (avoiding certain known `dplyr`/`dblyr` issues) (please see the side-notes of http://winvector.github.io/FluidData/partition_mutate.html for some references).

Value

.data with altered columns.

See Also

mutate_nse, mutate, mutate_at, :=

Examples

```
limit <- 3.5

datasets::iris %.>%
  mutate_se(., qae(Sepal_Long = Sepal.Length >= 2 * Sepal.Width,
                   Petal_Short := Petal.Length <= limit)) %.>%
  head(.)
```

---

mutate_seb *Run a sequence of quoted mutate blocks.*

Description

Run a sequence of quoted mutate blocks.

Usage

```
mutate_seb(d, blocks, env = parent.frame())
```

Arguments

d	data.frame to work on
blocks	list of sequence named char-array of mutate blocks
env	environment to work in.

Value

d with blocks applied in order

Examples

```
plan <- partition_mutate_qt(a1 := 1, b1 := a1, a2 := 2, b2 := a1 + a2)
print(plan)
d <- data.frame(x = 1) %.>% mutate_seb(., plan)
print(d)
```

---

novelName *Generate a name with a prefix disjoint from a set of names*

Description

Generate a name with a prefix disjoint from a set of names

Usage

```
novelName(prefix, names)
```

Arguments

prefix	character, desired prefix
names	character list of names to avoid

Value

new name disjoint from set of names

Examples

```
# basic op
novelName('b', c('a', 'b', 'c'))



# complex application (converting logistic
# links to probabilities).
```

```
d <- data.frame(
  exampleId = c(1, 1, 2, 2),
  resultLabel = c('a', 'b' , 'a', 'b'),
  linkValue = c(-5, 2, -2, -1),
  stringsAsFactors = FALSE)

totColName <- novelName('t', colnames(d))

d ->.;
  mutate_se(., c(totColName := "exp(linkValue)")) ->.;
  group_by_se(., "exampleId") ->.;
  mutate_se(., c("probability" :=
                 paste0(totColName, '/sum(', totColName, ')'))) ->.;
  deselect(., totColName)
```

`partition_mutate_qt` *Partition a sequence of mutate commands into longest ordered no create/use blocks.*

Description

We assume the sequence of expressions is in a valid order (all items available before use). This function partitions the expressions into ordered longest "no new value used blocks" by greedily scanning forward remaining expressions in order taking any that: have all their values available from earlier groups, do not use a value formed in the current group, and do not overwrite a value formed in the current group. For an example please see http://winvector.github.io/FluidData/partition_mutate.html.

Usage

```
partition_mutate_qt(...)
```

Arguments

...	mutate expressions with := used for assignment.

Details

Note: unlike `mutate_nse` `partition_mutate_qt` does not perform substitutions.

Value

ordered list of mutate_se assignment blocks

Examples

```
plan <- partition_mutate_qt(a1 := 1, b1 := a1, a2 := 2, b2 := a1 + a2)
print(plan)
d <- data.frame(x = 1) %.>% mutate_seb(., plan)
print(d)
```

`partition_mutate_se` *Partition a sequence of mutate commands into longest ordered no create/use blocks.*

Description

We assume the sequence of expressions is in a valid order (all items available before use). This function partitions the expressions into ordered longest "no new value used blocks" by greedily scanning forward remaining expressions in order taking any that: have all their values available from earlier groups, do not use a value formed in the current group, and do not overwrite a value formed in the current group. For an example please see http://winvector.github.io/FluidData/partition_mutate.html.

Usage

```
partition_mutate_se(exprs)
```

Arguments

`exprs`	list of source-text of a sequence of mutate expressions.

Value

ordered list of mutate_se assignment blocks

Examples

```
partition_mutate_se(c("a1" := "1", "b1" := "a1", "a2" := "2", "b2" := "a1 + a2"))
```

quote_mutate *Capture the expressions of a mutate-style command.*

Description

Capture the expressions of a mutate-style command.

Usage

```
quote_mutate(...)
```

Arguments

...	mutate expressions with := or = used for assignment.

Value

ordered list of mutate_se assignment blocks

Examples

```
assignments <- quote_mutate(a1 := 1, b1 = a1, a2 := 2, b2 := 7*(a1 + a2))
data.frame(x=1) %.>% mutate_se(., assignments)
```

rename_se *rename standard interface.*

Description

rename columns (much different syntax than rename_at). All left hand sides are new column names and all right hand sides are old column names (this allows swaps).

Usage

```
rename_se(.data, mapping, splitTerms = TRUE, env = parent.frame())
```

Arguments

.data	data.frame
mapping	named character vector of columns to rename (new names on the left, original names on the right; this may seem reversed but it matches dplyr::rename()).
splitTerms	logical, if TRUE into separate renames (if FALSE instead, pass all at once to dplyr).
env	environment to work in.

Details

Note: this method as the default setting `splitTerms = TRUE`, which is safer (avoiding certain known `dplyr`/`dblyr` issues) (please see the side-notes of http://winvector.github.io/FluidData/partition_mutate.html for some references).

Value

.data with renamed columns

See Also

`rename`, `rename_at`, `:=`

Examples

```
datasets::mtcars %.>%
   rename_se(., c("cylinders" := "cyl", "gears" := "gear")) %.>%
   head(.)
# # same as:
# datasets::mtcars %>%
#    rename(cylinders = cyl, gears = gear) %>%
#    head()

# rename_se allows column swaps
data.frame(a = 1, b = 2) %.>%
   rename_se(., c('a', 'b') := c('b', 'a'))
```

---

select_nse *Select columns non-standard (code capturing) interface.*

Description

Select column that are exactly the names captured unevaluated from . . . . This is to provide a simple interface that reliably uses non-standard captured names (and not consequences of further evaluation). Please see http://www.win-vector.com/blog/2018/09/a-subtle-flaw-in-some-popular-r-nse-interfaces for some discussion. Also accepts -name notation.

Usage

```
select_nse(.data, ...)
```

Arguments

`.data`	data frame or tbl to select columns from.
`...`	unevaluated symbols to use as column names.

Examples

```
y <- "x"

# returns y-column
dplyr::select(data.frame(x = 1, y = 2), y)

# returns x-column (very confusing!)
dplyr::select(data.frame(x = 1), y)

# returns y-column
select_nse(data.frame(x = 1, y = 2), y)

# throws when y is not the name of a column (good)
tryCatch(
  select_nse(data.frame(x = 1), y),
  error = function(e) { e }
)
```

`select_se` *Select columns standard interface.*

Description

Select columns. To remove columns please see `deselect`. Also accepts -column notation.

Usage

```
select_se(.data, colNames)
```

Arguments

`.data`	data.frame
`colNames`	character vector of columns to keep

Value

.data with only selected columns

See Also

`deselect`, `select`, `select_at`

Examples

```
datasets::mtcars %.>%
   select_se(., c("cyl", "gear")) %.>%
   head(.)
# essentially dplyr::select_at()

data.frame(a=1, b=2) %.>% select_se(., '-b')
```

`seplyr` *seplyr: Standard Evaluation Improved Interfaces for Common Data Manipulatio Tasks*

Description

The `seplyr` (standard evaluation `dplyr`) package supplies improved standard evaluation adapter methods for important common data manipulation tasks.

Details

In addition the `seplyr` package supplies several new "key operations bound together" methods. These include `group_summarize()` (which combines grouping, arranging and calculation in an atomic unit), `add_group_summaries()` (which joins grouped summaries into a `data.frame` in a well documented manner), `add_group_indices()` (which adds per-group identifiers to a `data.frame` without depending on row-order), `partition_mutate_qt()` (which optimizes mutate sequences), and `if_else_device()` (which simulates per-row if-else blocks in expression sequences).

`spread_se` *Collect values from blocks of rows into columns.*

Description

Standared inteface to `spread`. Take values from the columns named in the `columns` argument and move them into blocks of rows, placing values in the new column specified by `value` and indicating which column each value came from in the new column specified by `key`.

Usage

```
spread_se(data, key, value, ..., fill = NA, convert = FALSE,
  drop = TRUE, sep = NULL)
```

Arguments

data	data.frame to take values from.
key	character, name for existing column to get new column names from.
value	character, name for existing column to take values from.
...	not used, force later arguments to bind by name.
fill	passed to spread.
convert	passed to spread.
drop	passed to spread.
sep	passed to spread.

Value

converted data.

See Also

spread, gather_se

Examples

```
d <- wrapr::build_frame(
    'id', 'name_for_new_column' , 'value_to_take' |
    1   , 'col1'                , 'a'             |
    1   , 'col2'                , '10'            |
    2   , 'col1'                , 'b'             |
    2   , 'col2'                , '20'            )
spread_se(d,
   key = 'name_for_new_column',
   value = 'value_to_take')
```

---

summarize_nse *summarize non-standard evaluation interface.*

Description

summarize a data frame by the summarize terms from ....

Usage

```
summarize_nse(.data, ..., summarize_nse_warn = TRUE,
  env = parent.frame())

summarise_nse(.data, ..., summarize_nse_warn = TRUE,
  env = parent.frame())
```

Arguments

.data	data.frame
...	stringified expressions to summarize by.
summarize_nse_warn	logical, if TRUE warn about possible name collisions.
env	environment to work in.

Value

.data with summarized columns.

See Also

summarize_se, summarize, summarize_at, :=

Examples

```
datasets::iris %.>%
  summarize_nse(., Mean_Sepal_Length := mean(Sepal.Length),
                   Max_Sepal_Length := max(Sepal.Length))
```

---

summarize_se *summarize standard interface.*

Description

summarize a data frame by the summarizeTerms. Accepts arbitrary text as summarizeTerms to allow forms such as "mean(Sepal.Length)".

Usage

```
summarize_se(.data, summarizeTerms, ..., warn = TRUE,
  env = parent.frame())

summarise_se(.data, summarizeTerms, ..., warn = TRUE,
  env = parent.frame())
```

Arguments

.data	data.frame
summarizeTerms	character vector of column expressions to summarize by.
...	force later terms to be bound by name
warn	logical, if TRUE warn about possible name collisions.
env	environment to work in.

Value

.data with summarizeTerms summarization applied.

See Also

summarize, summarize_at, :=

Examples

```
# good
datasets::iris %.>%
  summarize_se(., qae(Mean_Sepal_Length := mean(Sepal.Length),
                      Max_Sepal_Length := max(Sepal.Length)))

# good
datasets::iris %.>%
  summarize_se(., qae(Sepal.Length := mean(Sepal.Length)))

# intentionally generates a warning
datasets::iris %.>%
  summarize_se(., qae(Sepal.Length := mean(Sepal.Length),
                      Max_Sepal_Length := max(Sepal.Length)))
```

---

tally_se *tally/count standard interface.*

Description

Add a new column named "n" with (optionally per-group) sums/counts.

Usage

```
tally_se(x, wt = NULL, sort = FALSE)
```

Arguments

x	data.frame to tally/count
wt	character optional column name containing row-weights (passed to count/tally)
sort	logical if TRUE sort result in descending order

Details

Note: dplyr::count, dplyr::add_count, dplyr::tally, and dplyr::add_tally are not S3 methods, so it may not be practical to re-dispatch seplyr calls to these dplyr implementations.

Value

.data with added column n, containing counts.

See Also

tally

Examples

```
datasets::mtcars %.>% tally_se(.)

datasets::mtcars %.>% tally_se(., wt = "cyl")
```

`transmute_nse` *transmute non-standard evaluation interface.*

Description

transmute a data frame by the transmuteterms from ....

Usage

```
transmute_nse(.data, ..., transmute_nse_env = parent.frame(),
  transmute_nse_warn = TRUE)
```

Arguments

`.data`	data.frame
`...`	stringified expressions to transmute by.
`transmute_nse_env`	environment to work in.
`transmute_nse_warn`	logical, if TRUE warn about possible name collisions.

Value

.data with altered columns(other columns dropped).

See Also

transmute_se, transmute, transmute_at, :=

Examples

```
datasets::iris %.>%
  transmute_nse(., Sepal_Long := Sepal.Length >= 2 * Sepal.Width,
                  Petal_Short := Petal.Length <= 3.5) %.>%
  summary(.)
```

`transmute_se` *transmute standard interface.*

Description

transmute a data frame by the transmuteTerms. Accepts arbitrary text as transmuteTerms to allow forms such as "Sepal.Length >= 2 * Sepal.Width".

Usage

```
transmute_se(.data, transmuteTerms, env = parent.frame(), warn = TRUE)
```

Arguments

`.data`	data.frame
`transmuteTerms`	character vector of column expressions to transmute by.
`env`	environment to work in.
`warn`	logical, if TRUE warn about possible name collisions.

Value

.data transumuted by transmuteTerms.

See Also

`transmute`, `transmute_at`, `:=`

Examples

```
datasets::iris %.>%
  transmute_se(., qae(Sepal_Long := Sepal.Length >= 2 * Sepal.Width,
                      Petal_Short := Petal.Length <= 3.5)) %.>%
  summary(.)
```

Index