Package 'blockr.dplyr'

May 7, 2026

Title Interactive 'dplyr' Data Transformation Blocks

Version 0.1.0

Description Extends 'blockr.core' with interactive blocks for visual data wrangling using 'dplyr' and 'tidyr' operations. Users can build data transformation pipelines through a graphical interface without writing code directly. Includes blocks for filtering, selecting, mutating, summarizing, joining, and arranging data, with support for complex expressions, grouping operations, and real-time validation.

URL https://bristolmyerssquibb.github.io/blockr.dplyr/

BugReports https://github.com/BristolMyersSquibb/blockr.dplyr/issues

License GPL (>= 3)

Depends R (>= 4.1.0)

Encoding UTF-8

RoxygenNote 7.3.3

Imports shiny, blockr.core (>= 0.1.1), dplyr, tidyr, shinyAce, glue, htmltools, bslib, jsonlite, shinyjs, utils, datasets

Suggests testthat (>= 3.0.0), knitr, rmarkdown, pkgdown

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

NeedsCompilation no

Author Christoph Sax [aut, cre] (ORCID: <https://orcid.org/0000-0002-7192-7044>), Nicolas Bennett [aut], David Granjon [aut], Mike Page [aut], Bristol Myers Squibb [fnd]

Maintainer Christoph Sax <christoph@cynkra.com>

Repository CRAN

Date/Publication 2025-12-18 14:30:18 UTC

Contents

`new_arrange_block` *Arrange block constructor*

Description

This block allows allows you to order the rows of a data frame by the values of selected columns (see `dplyr::arrange()`).

Usage

```
new_arrange_block(columns = character(), ...)
```

Arguments

`columns`	Columns to arrange by. Can be a character vector (ascending order) or a list of specifications with column and direction.
`...`	Forwarded to `blockr.core::new_block()`

Value

A transform block object of class `arrange_block`.

Examples

```
# Create an arrange block
new_arrange_block(columns = "mpg")

if (interactive()) {
  library(blockr.core)
  library(blockr.dplyr)

  # Basic usage - single column ascending
  serve(new_arrange_block(columns = "mpg"), list(data = mtcars))

  # Multiple columns with custom directions
  serve(
    new_arrange_block(
      columns = list(
        list(column = "cyl", direction = "asc"),
        list(column = "mpg", direction = "desc")
      )
    ),
    list(data = mtcars)
  )

  # Connected blocks - sort after categorizing
  serve(
    new_board(
      blocks = list(
        data = new_dataset_block(dataset = "mtcars"),
        categorized = new_mutate_block(
          exprs = list(
            car_type = paste0(
              "dplyr::case_when(cyl <= 4 ~ 'Economy', ",
              "cyl <= 6 ~ 'Standard', TRUE ~ 'Performance')"
            )
          )
        ),
        sorted = new_arrange_block(
          columns = list(
            list(column = "car_type", direction = "asc"),
            list(column = "mpg", direction = "desc"),
            list(column = "hp", direction = "desc")
          )
        )
      ),
      links = links(
        from = c("data", "categorized"),
        to = c("categorized", "sorted")
      )
    )
  )
}
```

`new_bind_cols_block` *Bind Columns Block Constructor*

Description

This block allows for column-wise combination of two or more data frames using `dplyr::bind_cols()`. It combines data frames side-by-side, requiring them to have the same number of rows. Duplicate column names are automatically handled by dplyr.

Usage

```
new_bind_cols_block(...)
```

Arguments

`...`	Forwarded to `blockr.core::new_block()`

Value

A block object for bind_cols operations

Examples

```
# Create a bind cols block
new_bind_cols_block()

if (interactive()) {
  library(blockr.core)
  library(blockr.dplyr)

  # Basic usage - combine different datasets horizontally
  serve(
    new_board(
      blocks = list(
        iris_data = new_dataset_block(dataset = "iris"),
        mtcars_data = new_dataset_block(dataset = "mtcars"),
        head1 = new_slice_block(type = "head", n = 5),
        head2 = new_slice_block(type = "head", n = 5),
        combined = new_bind_cols_block()
      ),
      links = links(
        from = c("iris_data", "mtcars_data", "head1", "head2"),
        to = c("head1", "head2", "combined", "combined"),
        input = c("data", "data", "1", "2")
      )
    )
  )

  # Combine selected columns from same dataset
  serve(
```

```
    new_board(
      blocks = list(
        mtcars_data = new_dataset_block(dataset = "mtcars"),
        engine_cols = new_select_block(columns = c("mpg", "cyl", "hp")),
        weight_cols = new_select_block(columns = c("wt", "qsec")),
        combined = new_bind_cols_block()
      ),
      links = links(
        from = c("mtcars_data", "mtcars_data", "engine_cols", "weight_cols"),
        to = c("engine_cols", "weight_cols", "combined", "combined"),
        input = c("data", "data", "1", "2")
      )
    )
  )
}
```

`new_bind_rows_block` *Bind Rows Block Constructor*

Description

This block allows for row-wise combination of two or more data frames using `dplyr::bind_rows()`. It stacks data frames vertically, matching columns by name and filling missing columns with NA values.

Usage

```
new_bind_rows_block(id_name = "", ...)
```

Arguments

`id_name`	Character string, name for the ID column. If non-empty, adds a column identifying source data frames. Default "" (disabled).
`...`	Forwarded to `blockr.core::new_block()`

Value

A block object for bind_rows operations

Examples

```
# Create a bind rows block
new_bind_rows_block()

if (interactive()) {
  library(blockr.core)
  library(blockr.dplyr)
```

```
# Basic usage - stack filtered datasets
serve(
  new_board(
    blocks = list(
      iris_data = new_dataset_block(dataset = "iris"),
      setosa = new_filter_expr_block(exprs = list("Species == 'setosa'")),
      versicolor = new_filter_expr_block(exprs = list("Species == 'versicolor'")),
      combined = new_bind_rows_block()
    ),
    links = links(
      from = c("iris_data", "iris_data", "setosa", "versicolor"),
      to = c("setosa", "versicolor", "combined", "combined"),
      input = c("data", "data", "1", "2")
    )
  )
)

# With ID column to track source
serve(
  new_board(
    blocks = list(
      iris_data = new_dataset_block(dataset = "iris"),
      setosa = new_filter_expr_block(exprs = list("Species == 'setosa'")),
      versicolor = new_filter_expr_block(exprs = list("Species == 'versicolor'")),
      combined = new_bind_rows_block(id_name = "source")
    ),
    links = links(
      from = c("iris_data", "iris_data", "setosa", "versicolor"),
      to = c("setosa", "versicolor", "combined", "combined"),
      input = c("data", "data", "1", "2")
    )
  )
)
}
```

`new_filter_block` *Filter block constructor*

Description

This block allows filtering rows in a data frame by selecting specific values from columns (see `dplyr::filter()`). Provides a visual interface where users can select columns and choose which values to include or exclude without writing R expressions. Supports multiple conditions with AND/OR logic.

Usage

```
new_filter_block(conditions = list(), preserve_order = FALSE, ...)
```

Arguments

conditions	List of filter conditions. Each condition should be a list with elements: column (character), values (vector), mode ("include" or "exclude"), and optionally operator ("&" or "|") specifying how this condition connects to the previous one
preserve_order	Logical. If TRUE, preserves the order of selected values in the filtered output (default: FALSE)
...	Additional arguments forwarded to blockr.core::new_block()

Details

For expression-based filtering, see new_filter_expr_block().

Value

A block object for value-based filter operations

See Also

blockr.core::new_transform_block()

Examples

```
# Create a filter block
new_filter_block()

if (interactive()) {
  # Basic usage with mtcars dataset
  library(blockr.core)
  serve(new_filter_block(), list(data = mtcars))

  # With initial condition
  serve(new_filter_block(
    conditions = list(
      list(column = "cyl", values = c(4, 6), mode = "include")
    )
  ), list(data = mtcars))

  # Connected blocks example
  serve(
    new_board(
      blocks = list(
        a = new_dataset_block(),
        b = new_filter_block()
      ),
      links = links(
        from = c("a"),
        to = c("b")
      )
    )
  )
}
```

new_filter_expr_block *Expression filter block constructor*

Description

This block allows filtering rows in a data frame based on R expressions (see dplyr::filter()). Supports multiple conditions with AND/OR logic. Changes are applied after clicking the submit button.

Usage

```
new_filter_expr_block(exprs = "TRUE", ...)
```

Arguments

exprs	Reactive expression returning character vector of filter conditions (default: "TRUE" for no filtering)
...	Additional arguments forwarded to blockr.core::new_block()

Value

A block object for expression-based filter operations

See Also

blockr.core::new_transform_block()

Examples

```
# Create a filter block
new_filter_expr_block("mpg > 20")

if (interactive()) {
  # Basic usage with mtcars dataset
  library(blockr.core)
  serve(new_filter_expr_block(), list(data = mtcars))

  # With custom initial condition
  serve(new_filter_expr_block("mpg > 20"), list(data = mtcars))

  # Connected blocks example
  serve(
    new_board(
      blocks = list(
        a = new_dataset_block(),
        b = new_filter_expr_block()
      ),
      links = links(
        from = c("a"),
```

```
        to = c("b")
      )
    )
  )
}
```

`new_join_block` *Join block constructor*

Description

This block allows for joining of two `data.frame` objects with advanced multi-column support including same-name and different-name joins (see `dplyr::left_join()`, `dplyr::inner_join()`, etc.).

Usage

```
new_join_block(type = character(), by = character(), ...)
```

Arguments

`type`	Join type (left_join, inner_join, right_join, full_join, semi_join, anti_join)
`by`	Column(s) to join on - can be character vector for same-name joins or named list for different-name joins
`...`	Forwarded to `blockr.core::new_block()`

Value

A transform block object of class `join_block`.

Examples

```
# Create a join block
new_join_block(type = "left_join")

if (interactive()) {
  library(blockr.core)
  library(blockr.dplyr)

  # Basic left join - automatically detects common columns
  serve(
    new_board(
      blocks = list(
        data1 = new_dataset_block(dataset = "band_members"),
        data2 = new_dataset_block(dataset = "band_instruments"),
        joined = new_join_block(type = "left_join")
      ),
      links = links(
        from = c("data1", "data2"),
```

```
      to = c("joined", "joined"),
      input = c("x", "y")
    )
  )
)

# Inner join with explicit join column
serve(
  new_board(
    blocks = list(
      data1 = new_dataset_block(dataset = "band_members"),
      data2 = new_dataset_block(dataset = "band_instruments"),
      joined = new_join_block(type = "inner_join", by = "name")
    ),
    links = links(
      from = c("data1", "data2"),
      to = c("joined", "joined"),
      input = c("x", "y")
    )
  )
)

# Right join - keep all rows from right dataset
serve(
  new_board(
    blocks = list(
      data1 = new_dataset_block(dataset = "band_members"),
      data2 = new_dataset_block(dataset = "band_instruments"),
      joined = new_join_block(type = "right_join")
    ),
    links = links(
      from = c("data1", "data2"),
      to = c("joined", "joined"),
      input = c("x", "y")
    )
  )
)

# Anti join - find rows in left without matches in right
serve(
  new_board(
    blocks = list(
      data1 = new_dataset_block(dataset = "band_members"),
      data2 = new_dataset_block(dataset = "band_instruments"),
      joined = new_join_block(type = "anti_join")
    ),
    links = links(
      from = c("data1", "data2"),
      to = c("joined", "joined"),
      input = c("x", "y")
    )
  )
)
```

```
}
```

`new_mutate_expr_block` *Mutate expression block constructor*

Description

This block allows to add new variables and preserve existing ones using R expressions (see `dplyr::mutate()`). Changes are applied after clicking the submit button.

Usage

```
new_mutate_expr_block(exprs = list(new_col = "1"), by = character(), ...)
```

Arguments

`exprs`	Reactive expression returning character vector of expressions
`by`	Character vector of column names for grouping. Default is empty.
`...`	Additional arguments forwarded to `blockr.core::new_block()`

Value

A block object for mutate operations

See Also

`blockr.core::new_transform_block()`

Examples

```
# Create a mutate expression block
new_mutate_expr_block(exprs = list(mpg_squared = "mpg^2"))

if (interactive()) {
  # Basic usage with mtcars datase
  library(blockr.core)
  serve(new_mutate_expr_block(), data = list(data = mtcars))

  # With a custom dataset
  df <- data.frame(x = 1:5, check.names = FALSE)
  df$`2025 Sales` <- letters[1:5]
  serve(new_mutate_expr_block(), data = list(data = df))
}
```

new_pivot_longer_block *Pivot Longer block constructor*

Description

This block reshapes data from wide to long format by pivoting multiple columns into two columns: one containing the original column names and another containing the values (see `tidyr::pivot_longer()`).

Usage

```
new_pivot_longer_block(
  cols = character(),
  names_to = "name",
  values_to = "value",
  values_drop_na = FALSE,
  names_prefix = "",
  ...
)
```

Arguments

`cols`	Character vector of column names to pivot into longer format. If empty, all columns will be available for selection.
`names_to`	Name of the new column to create from the column names. Default is "name".
`values_to`	Name of the new column to create from the values. Default is "value".
`values_drop_na`	If TRUE, rows with NA values will be dropped. Default is FALSE.
`names_prefix`	Optional prefix to remove from column names before storing in the names_to column. For example, "col_" would remove that prefix from column names like "col_a", "col_b".
`...`	Additional arguments forwarded to `blockr.core::new_transform_block()`

Value

A block object for pivot_longer operations

See Also

`blockr.core::new_transform_block()`, `tidyr::pivot_longer()`

Examples

```
# Create a pivot longer block
new_pivot_longer_block()

if (interactive()) {
```

```
  # Basic usage with wide format data
  library(blockr.core)
  wide_data <- data.frame(
    id = 1:3,
    measurement_a = c(10, 20, 30),
    measurement_b = c(15, 25, 35),
    measurement_c = c(12, 22, 32)
  )
  serve(
    new_pivot_longer_block(
      cols = c("measurement_a", "measurement_b", "measurement_c"),
      names_to = "measurement_type",
      values_to = "value"
    ),
    data = list(data = wide_data)
  )

  # With names_prefix to clean column names
  serve(
    new_pivot_longer_block(
      cols = c("measurement_a", "measurement_b", "measurement_c"),
      names_to = "type",
      values_to = "measurement",
      names_prefix = "measurement_"
    ),
    data = list(data = wide_data)
  )
}
```

`new_pivot_wider_block` *Pivot Wider block constructor*

Description

This block reshapes data from long to wide format by pivoting column values into new columns (see `tidyr::pivot_wider()`). This is the inverse operation of pivot_longer.

Usage

```
new_pivot_wider_block(
  names_from = character(),
  values_from = character(),
  id_cols = character(),
  values_fill = "",
  names_sep = "_",
  names_prefix = "",
  ...
)
```

Arguments

names_from	Character vector specifying which column(s) to use for new column names. Can be a single column or multiple columns.
values_from	Character vector specifying which column(s) to use for cell values. Can be a single column or multiple columns.
id_cols	Character vector of columns that uniquely identify each row. If empty (default), uses all columns not specified in names_from or values_from.
values_fill	Optional value to use for missing combinations. Can be a single value like "0" or "NA". Leave empty to keep missing values as NA.
names_sep	Separator to use when names_from specifies multiple columns. Default is "_".
names_prefix	Optional prefix to add to all new column names.
...	Additional arguments forwarded to blockr.core::new_transform_block()

Value

A block object for pivot_wider operations

See Also

blockr.core::new_transform_block(), tidyr::pivot_wider()

Examples

```
# Create a pivot wider block
new_pivot_wider_block()

if (interactive()) {
  # Basic usage with long format data
  library(blockr.core)
  long_data <- data.frame(
    id = rep(1:3, each = 3),
    measurement_type = rep(c("a", "b", "c"), 3),
    value = c(10, 15, 12, 20, 25, 22, 30, 35, 32)
  )
  serve(
    new_pivot_wider_block(
      names_from = "measurement_type",
      values_from = "value"
    ),
    data = list(data = long_data)
  )

  # With values_fill to replace NAs
  serve(
    new_pivot_wider_block(
      names_from = "measurement_type",
      values_from = "value",
      values_fill = "0"
    ),
```

```
    data = list(data = long_data)
  )

  # With custom names_prefix
  serve(
    new_pivot_wider_block(
      names_from = "measurement_type",
      values_from = "value",
      names_prefix = "measure_"
    ),
    data = list(data = long_data)
  )
}
```

`new_rename_block` *Rename block constructor*

Description

This block allows renaming columns in a data frame using the visual interface (see `dplyr::rename()`). Changes are applied after clicking the submit button. Uses new_name = old_name syntax where new_name is what you want to call the column and old_name is the current column name.

Usage

```
new_rename_block(renames = list(new_col = character()), ...)
```

Arguments

`renames`	Named list or vector of renames in new_name = old_name format
`...`	Additional arguments forwarded to `blockr.core::new_block()`

Value

A block object for rename operations

See Also

`blockr.core::new_transform_block()`

Examples

```
# Create a rename block
new_rename_block(list(miles_per_gallon = "mpg", cylinders = "cyl"))

if (interactive()) {
  # Basic usage with mtcars dataset
  library(blockr.core)
  serve(new_rename_block(), data = list(data = mtcars))
```

```
  # With predefined renames
  serve(
    new_rename_block(list(miles_per_gallon = "mpg", cylinders = "cyl")),
    data = list(data = mtcars)
  )

  # Connected blocks example
  serve(
    new_board(
      blocks = list(
        a = new_dataset_block(),
        b = new_rename_block(list(horsepower = "hp"))
      ),
      links = links(
        from = c("a"),
        to = c("b")
      )
    )
  )
}
```

`new_select_block` *Select block constructor*

Description

This block allows performing column subsetting on `data.frame` objects (see `dplyr::select()`). Columns can be selected and reordered by dragging, and an exclude mode allows for negative selection using dplyr's minus syntax. Optionally, distinct rows can be kept after selection.

Usage

```
new_select_block(columns = character(), exclude = FALSE, distinct = FALSE, ...)
```

Arguments

`columns`	Selected columns (character vector). If empty, selects all columns.
`exclude`	Logical. If TRUE, uses exclude mode (dplyr minus syntax: `-c(col1, col2)`). If FALSE (default), uses include mode (selects specified columns).
`distinct`	Logical. If TRUE, keeps only distinct/unique rows after selecting columns. If FALSE (default), returns all rows. This replaces the old `new_distinct_block()` functionality.
`...`	Forwarded to `blockr.core::new_transform_block()`

Details

Note: This block replaces the deprecated `new_distinct_block()`. Use the `distinct` parameter to get unique rows after column selection.

The select block provides a sortable multi-select interface where columns can be:

- Selected/deselected by clicking
- Reordered by dragging (order is preserved in output)
- Removed individually using the × button

Include mode (exclude = FALSE, default):

- Selected columns are included in output
- Empty selection = select all (`select(data, dplyr::everything())`)

Exclude mode (exclude = TRUE):

- Selected columns are excluded from output using minus syntax
- Empty selection = select all (`select(data)`)
- Efficient for large datasets when you want to remove just a few columns

Distinct mode (distinct = TRUE):

- Keeps only distinct rows after column selection
- Equivalent to piping `select()` output to `distinct()`
- Useful for finding unique combinations of selected columns

Value

A transform block object of class `select_block`.

Examples

```
# Create a select block
new_select_block(columns = c("mpg", "cyl", "hp"))

if (interactive()) {
  # Basic usage with mtcars dataset
  library(blockr.core)
  serve(new_select_block(), list(data = mtcars))

  # With initial column selection
  serve(new_select_block(columns = c("mpg", "cyl", "hp")), list(data = mtcars))

  # Exclude mode (select all except specified columns)
 serve(new_select_block(columns = c("gear", "carb"), exclude = TRUE), list(data = mtcars))

  # Select with distinct (unique combinations)
 serve(new_select_block(columns = c("cyl", "gear"), distinct = TRUE), list(data = mtcars))

  # Full deduplication (distinct on all columns)
```

```
serve(new_select_block(distinct = TRUE), list(data = mtcars))

# Connected blocks example
serve(
  new_board(
    blocks = list(
      a = new_dataset_block(),
      b = new_select_block()
    ),
    links = links(
      from = c("a"),
      to = c("b")
    )
  )
)
}
```

`new_separate_block` *Separate block constructor*

Description

This block separates a single character column into multiple columns by splitting on a separator pattern (see `tidyr::separate()`). This is the inverse operation of unite.

Usage

```
new_separate_block(
  col = character(),
  into = c("col1", "col2"),
  sep = "[^[:alnum:]]+",
  remove = TRUE,
  convert = FALSE,
  extra = "warn",
  fill = "warn",
  ...
)
```

Arguments

`col`	Character string specifying which column to separate. If empty (default), all columns will be available for selection.
`into`	Character vector of names for the new columns. Can be specified as a character vector or a comma-separated string (e.g., "col1, col2, col3"). Default is c("col1", "col2").
`sep`	Separator between columns. Can be a regular expression or numeric positions. Default is `"[^[:alnum:]]+"` (any non-alphanumeric character).
`remove`	If TRUE (default), remove input column from output data frame.

convert	If TRUE, will run type.convert() with as.is = TRUE on new columns. Default is FALSE.
extra	How to handle extra pieces when there are too many: "warn" (default), "drop", or "merge".
fill	How to handle missing pieces when there are too few: "warn" (default), "right", or "left".
...	Additional arguments forwarded to blockr.core::new_transform_block()

Value

A block object for separate operations

See Also

blockr.core::new_transform_block(), tidyr::separate()

Examples

```
# Create a separate block
new_separate_block()

if (interactive()) {
  # Basic usage - separate full name into first and last
  library(blockr.core)
  people_data <- data.frame(
    full_name = c("John Doe", "Jane Smith", "Bob Johnson"),
    age = c(30, 25, 35)
  )
  serve(
    new_separate_block(
      col = "full_name",
      into = c("first_name", "last_name"),
      sep = " "
    ),
    data = list(data = people_data)
  )

  # Separate date components
  date_data <- data.frame(
    date_string = c("2024-01-15", "2024-02-20", "2024-03-25")
  )
  serve(
    new_separate_block(
      col = "date_string",
      into = c("year", "month", "day"),
      sep = "-",
      convert = TRUE
    ),
    data = list(data = date_data)
  )
```

```
  # Using regex separator
  mixed_data <- data.frame(
    mixed_col = c("a-b", "c_d", "e.f")
  )
  serve(
    new_separate_block(
      col = "mixed_col",
      into = c("col1", "col2"),
      sep = "[-_.]"
    ),
    data = list(data = mixed_data)
  )
}
```

`new_slice_block` *Slice block constructor*

Description

This block allows row selection using various dplyr slice functions (see `dplyr::slice()`, `dplyr::slice_head()`, `dplyr::slice_tail()`, `dplyr::slice_min()`, `dplyr::slice_max()`, `dplyr::slice_sample()`). Features reactive UI with immediate updates and comprehensive grouping support.

Usage

```
new_slice_block(
  type = "head",
  n = 5,
  prop = NULL,
  order_by = character(),
  with_ties = TRUE,
  weight_by = character(),
  replace = FALSE,
  rows = "1:5",
  by = character(),
  ...
)
```

Arguments

`type`	Character string specifying slice type: "head", "tail", "min", "max", "sample", or "custom"
`n`	Number of rows to select (default: 5). Mutually exclusive with prop.
`prop`	Proportion of rows to select (0 to 1, default: NULL). When specified, n is ignored.
`order_by`	Column name to order by (for slice_min/slice_max)
`with_ties`	Logical, whether to include ties (for slice_min/slice_max)

weight_by	Column name for weighted sampling (for slice_sample)
replace	Logical, whether to sample with replacement (for slice_sample)
rows	Custom row positions (for slice)
by	Character vector of column names for grouping
...	Additional arguments forwarded to blockr.core::new_block()

Value

A block object for slice operations

See Also

blockr.core::new_transform_block()

Examples

```
# Create a slice block
new_slice_block(type = "head", n = 5)

if (interactive()) {
  # Basic usage
  library(blockr.core)
  serve(new_slice_block(), list(data = mtcars))

  # Select first 5 rows
  serve(new_slice_block(type = "head", n = 5), list(data = mtcars))

  # Select rows with highest mpg values
  serve(new_slice_block(type = "max", order_by = "mpg", n = 3), list(data = mtcars))

  # Random sampling
  serve(new_slice_block(type = "sample", n = 10, replace = FALSE), list(data = mtcars))
}
```

---

new_summarize_block *Summarize block constructor*

Description

This block provides a no-code interface for summarizing data (see dplyr::summarize()). Instead of writing expressions, users select summary functions from dropdowns (mean, median, sum, etc.), choose columns to summarize, and specify new column names.

Usage

```
new_summarize_block(
  summaries = list(count = list(func = "dplyr::n", col = "")),
  by = character(),
  ...
)
```

Arguments

summaries	Named list where each element is a list with 'func' and 'col' elements. For example: list(avg_mpg = list(func = "mean", col = "mpg"))
by	Columns to define grouping
...	Additional arguments forwarded to blockr.core::new_block()

Details

For expression-based summarization, see new_summarize_expr_block().

Value

A block object for no-code summarize operations

Extending available functions

The list of available summary functions can be extended using the blockr.dplyr.summary_functions option. Set this option to a named character vector where names are display labels and values are function calls:

```
options(
  blockr.dplyr.summary_functions = c(
    "extract parentheses (paren_num)" = "blockr.topline::paren_num"
  )
)
```

If a description is not provided (empty name), the function name will be used as the display label.

See Also

blockr.core::new_transform_block(), new_summarize_expr_block()

Examples

```
# Create a summarize block
new_summarize_block()

if (interactive()) {
  # Basic usage with mtcars dataset
  library(blockr.core)
  serve(new_summarize_block(), data = list(data = mtcars))

  # With predefined summaries
  serve(
    new_summarize_block(
      summaries = list(
        avg_mpg = list(func = "mean", col = "mpg"),
        max_hp = list(func = "max", col = "hp")
      )
    ),
```

```
    data = list(data = mtcars)
  )

  # With grouping
  serve(
    new_summarize_block(
      summaries = list(avg_mpg = list(func = "mean", col = "mpg")),
      by = "cyl"
    ),
    data = list(data = mtcars)
  )
}
```

`new_summarize_expr_block` *Summarize expression block constructor*

Description

This block allows to add new variables by summarizing over groups using R expressions (see `dplyr::summarize()`). Changes are applied after clicking the submit button.

Usage

```
new_summarize_expr_block(
  exprs = list(count = "dplyr::n()"),
  by = character(),
  unpack = FALSE,
  ...
)
```

Arguments

`exprs`	Reactive expression returning character vector of expressions
`by`	Columns to define grouping
`unpack`	Logical flag to unpack data frame columns from helper functions. When `TRUE`, expressions that return data frames will have their columns unpacked into separate columns. When `FALSE`, data frames are kept as nested list-columns. Default is `FALSE`.
`...`	Additional arguments forwarded to `blockr.core::new_block()`

Details

For no-code summarization using dropdowns, see `new_summarize_block()`.

Value

A block object for summarize operations

Unpacking Helper Function Results

When unpack = TRUE, helper functions that return data frames will have their columns unpacked into separate columns in the result. This is useful for helper functions like stat_label() that return multiple statistics as columns.

```
# Without unpacking (default)
new_summarize_expr_block(
  exprs = list(stats = "helper_func(...)"),
  unpack = FALSE
)
# Result: Creates nested list-column "stats" containing the data frame

# With unpacking
new_summarize_expr_block(
  exprs = list(stats = "helper_func(...)"),
  unpack = TRUE
)
# Result: Columns from helper_func() are unpacked into separate columns
```

See Also

blockr.core::new_transform_block()

Examples

```
# Create a summarize expression block
new_summarize_expr_block()

if (interactive()) {
  # Basic usage with mtcars dataset
  library(blockr.core)
  serve(new_summarize_expr_block(), list(data = mtcars))

  # With a custom dataset
  df <- data.frame(x = 1:5, y = letters[1:5])
  serve(new_summarize_expr_block(), list(data = df))

  # Using unpack to expand helper function results
  # Define the helper in your environment first
  calc_stats <- function(df) {
    data.frame(
      mean_x = mean(df$x),
      mean_y = mean(df$y),
      sum_x = sum(df$x),
      sum_y = sum(df$y)
    )
  }

  # With unpacking enabled
  serve(
```

```
    new_summarize_expr_block(
      exprs = list(stats = "calc_stats(pick(everything()))"),
      by = "group",
      unpack = TRUE
    ),
    list(data = data.frame(x = 1:6, y = 10:15, group = rep(c("A", "B"), 3)))
  )
  # Result: group, mean_x, mean_y, sum_x, sum_y (columns unpacked)
}
```

`new_unite_block` *Unite block constructor*

Description

This block combines multiple columns into a single column by pasting their values together (see `tidyr::unite()`). This is useful for creating composite identifiers or labels from multiple fields.

Usage

```
new_unite_block(
  col = "united",
  cols = character(),
  sep = "_",
  remove = TRUE,
  na.rm = FALSE,
  ...
)
```

Arguments

`col`	Name for the new united column. Default is "united".
`cols`	Character vector of column names to unite together. If empty (default), all columns will be available for selection.
`sep`	Separator to use between values. Default is "_".
`remove`	If TRUE (default), remove input columns from output data frame.
`na.rm`	If TRUE, missing values will be removed prior to uniting each row. Default is FALSE.
`...`	Additional arguments forwarded to `blockr.core::new_transform_block()`

Value

A block object for unite operations

See Also

`blockr.core::new_transform_block()`, `tidyr::unite()`

Examples

```
# Create a unite block
new_unite_block()

if (interactive()) {
  # Basic usage - combine first and last name
  library(blockr.core)
  people_data <- data.frame(
    first_name = c("John", "Jane", "Bob"),
    last_name = c("Doe", "Smith", "Johnson"),
    age = c(30, 25, 35)
  )
  serve(
    new_unite_block(
      col = "full_name",
      cols = c("first_name", "last_name"),
      sep = " "
    ),
    data = list(data = people_data)
  )

  # With custom separator
  serve(
    new_unite_block(
      col = "id",
      cols = c("first_name", "last_name"),
      sep = "-",
      remove = TRUE
    ),
    data = list(data = people_data)
  )

  # With NA removal
  data_with_na <- data.frame(
    prefix = c("Dr.", NA, "Prof."),
    first = c("John", "Jane", "Bob"),
    last = c("Doe", "Smith", "Johnson")
  )
  serve(
    new_unite_block(
      col = "full_name",
      cols = c("prefix", "first", "last"),
      sep = " ",
      na.rm = TRUE
    ),
    data = list(data = data_with_na)
  )
}
```

Index