Package: dsTidyverseClient (via r-universe)

November 11, 2024

Type Package

Title 'DataSHIELD' 'Tidyverse' Clientside Package

Version 1.0.0

Maintainer Tim Cadman <t.j.cadman@umcg.nl>

Description Implementation of selected 'Tidyverse' functions within 'DataSHIELD', an open-source federated analysis solution in R. Currently, 'DataSHIELD' contains very limited tools for data manipulation, so the aim of this package is to improve the researcher experience by implementing essential functions for data manipulation, including subsetting, filtering, grouping, and renaming variables. This is the clientside package which should be installed locally, and is used in conjuncture with the serverside package 'dsTidyverse' which is installed on the remote server holding the data. For more information, see <https://www.tidyverse.org/>, <https://datashield.org/> and <https://github.com/molgenis/ds-tidyverse>.

License LGPL (>= 2.1)

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Imports DSI (>= 1.7.0), cli, rlang, assertthat

Suggests testthat (>= 3.0.0), knitr, rmarkdown, DSLite, dsBase, dsBaseClient, dsTidyverse, dplyr

Config/testthat/edition 3

Additional_repositories https://cran.obiba.org/

VignetteBuilder knitr

Repository https://molgenis.r-universe.dev

RemoteUrl https://github.com/molgenis/ds-tidyverse-client

RemoteRef HEAD

RemoteSha b9b5fb310d10d3cd0e6ad29f1d4d4cb50da489f2

Contents

`ds.arrange` *Order the rows of a data frame by the values of selected columns*

Description

DataSHIELD implentation of `dplyr::arrange`.

Usage

```
ds.arrange(
  df.name = NULL,
  tidy_expr = NULL,
  .by_group = NULL,
  newobj = NULL,
  datasources = NULL
)
```

Arguments

`df.name`	Character specifying a serverside data frame or tibble.
`tidy_expr`	A list containing variables, or functions of variables. Use `dplyr::desc()` to sort a variable in descending order.
`.by_group`	If TRUE, will sort first by grouping variable. Applies to grouped data frames only.
`newobj`	Character specifying name for new server-side data frame.
`datasources`	DataSHIELD connections object.

Value

No return value, called for its side effects. An object (typically a data frame or tibble) with the name specified by `newobj` is created on the server.

Examples

```
## Not run:
ds.arrange(
  df.name = "mtcars",
  tidy_expr = list(drat),
  newobj = "sorted_df",
  datasources = conns
)

## End(Not run)
```

---

ds.as_tibble *Coerce a data frame or matrix to a tibble.*

Description

DataSHIELD implementation of `tibble::as_tibble`. Currently only implemented for data frames and tibbles.

Usage

```
ds.as_tibble(
  x = NULL,
  .rows = NULL,
  .name_repair = "check_unique",
  rownames = NULL,
  newobj = NULL,
  datasources = NULL
)
```

Arguments

`x`	A data frame or matrix.
`.rows`	The number of rows, useful to create a 0-column tibble or just as an additional check.
`.name_repair`	Treatment of problematic column names: • "minimal": No name repair or checks, beyond basic existence. • "unique": Make sure names are unique and not empty. • "check_unique": (default value), no name repair, but check they are unique. • "universal": Make the names unique and syntactic.
`rownames`	How to treat existing row names of a data frame or matrix:

- 'NULL': remove row names. This is the default.
- 'NA': keep row names.
- A string: the name of a new column. Existing rownames are transferred into this column and the `row.names` attribute is deleted. No name repair is applied to the new column name, even if 'x' already contains a column of that name.

`newobj` Character specifying name for new server-side data frame.

`datasources` DataSHIELD connections object.

Value

No return value, called for its side effects. A tibble with the name specified by `newobj` is created on the server.

Examples

```
## Not run:
ds.as_tibble(
  x = "mtcars",
  newobj = "mtcars_tib",
  datasources = conns
)

## End(Not run)
```

---

ds.bind_cols *Bind multiple data frames by column*

Description

DataSHIELD implementation of `dplyr::bind_cols`.

Usage

```
ds.bind_cols(
  to_combine = NULL,
  .name_repair = c("unique", "universal", "check_unique", "minimal"),
  newobj = NULL,
  datasources = NULL
)
```

Arguments

`to_combine` Data frames to combine. Each argument can either be a data frame, a list that could be a data frame, or a list of data frames. Columns are matched by name, and any missing columns will be filled with NA.

.name_repair	One of "unique", "universal", or "check_unique". See vctrs::vec_as_names() for the meaning of these options.
newobj	Character specifying name for new server-side data frame.
datasources	datashield connections object.

Value

No return value, called for its side effects. A data frame with the name specified by newobj and the same type as the first element of to_combine is created on the server.

Examples

```
## Not run:
## First log in to a DataSHIELD session with mtcars dataset loaded.

ds.bind_cols(
  to_combine = list(mtcars, mtcars),
  .name_repair = "universal",
  newobj = "test",
  datasources = conns
)

## Refer to the package vignette for more examples.

## End(Not run)
```

---

ds.bind_rows *Bind multiple data frames by row.*

Description

DataSHIELD implementation of dplyr::bind_rows.

Usage

```
ds.bind_rows(to_combine = NULL, .id = NULL, newobj = NULL, datasources = NULL)
```

Arguments

to_combine	Data frames to combine. Each argument can either be a data frame, a list that could be a data frame, or a list of data frames. Columns are matched by name, and any missing columns will be filled with NA.
.id	The name of an optional identifier column. Provide a string to create an output column that identifies each input. The column will use names if available, otherwise it will use positions.
newobj	Character specifying name for new server-side data frame.
datasources	datashield connections object.

Value

No return value, called for its side effects. A data frame with the name specified by `newobj` and the same type as the first element of `to_combine` is created on the server.

Examples

```
## Not run:
## First log in to a DataSHIELD session with mtcars dataset loaded.

ds.bind_rows(
  to_combine = list(mtcars, mtcars),
  newobj = "test",
  datasources = conns
)

## Refer to the package vignette for more examples.

## End(Not run)
```

---

ds.case_when *A general vectorised if-else*

Description

DataSHIELD implementation of `dplyr::case_when`.

Usage

```
ds.case_when(
  tidy_expr = NULL,
  .default = NULL,
  .ptype = NULL,
  .size = NULL,
  newobj = NULL,
  datasources = NULL
)
```

Arguments

`tidy_expr` A list containing a sequence of two-sided formulas:

- The left hand side (LHS) determines which values match this case.
- The right hand side (RHS) provides the replacement value.
- The LHS inputs must evaluate to logical vectors.
- The RHS inputs will be coerced to their common type.

All inputs will be recycled to their common size. We encourage all LHS inputs to be the same size. Recycling is mainly useful for RHS inputs, where you might supply a size 1 input that will be recycled to the size of the LHS inputs. NULL inputs are ignored.

.default	The value used when all of the LHS inputs return either FALSE or NA.
.ptype	An optional prototype declaring the desired output type. If supplied, this overrides the common type of true, false, and missing.
.size	An optional size declaring the desired output size. If supplied, this overrides the size of condition.
newobj	Character specifying name for new server-side data frame.
datasources	datashield connections object.

Value

No return value, called for its side effects. A vector with the same size as the common size computed from the inputs in `tidy_expr` and the same type as the common type of the RHS inputs in `tidy_expr` is created on the server.

Examples

```
## Not run:
## First log in to a DataSHIELD session with mtcars dataset loaded.

ds.case_when(
  tidy_expr = list(
    mtcars$mpg < 10 ~ "low",
    mtcars$mpg >= 10 & mtcars$mpg < 20 ~ "medium",
    mtcars$mpg >= 20 ~ "high"
  ),
  newobj = "test",
  datasources = conns
)

## Refer to the package vignette for more examples.

## End(Not run)
```

---

ds.distinct *Keep distinct/unique rows*

Description

DataSHIELD implentation of `dplyr::distinct`.

Usage

```
ds.distinct(
  df.name = NULL,
  tidy_expr = NULL,
  .keep_all = FALSE,
  newobj = NULL,
  datasources = NULL
)
```

Arguments

df.name	Character specifying a serverside data frame or tibble.
tidy_expr	Optionally, list of variables to use when determining uniqueness. If there are multiple rows for a given combination of inputs, only the first row will be preserved. If omitted, will use all variables in the data frame.
.keep_all	If TRUE, keep all variables in .data. If a combination of expr is not distinct, this keeps the first row of values.
newobj	Character specifying name for new server-side data frame.
datasources	DataSHIELD connections object.

Value

No return value, called for its side effects. An object (typically a data frame or tibble) with the name specified by newobj is created on the server.

Examples

```
## Not run:
ds.distinct(
  df.name = "mtcars",
  expr = list(mpg, cyl),
  newobj = "distinct_df"
)

## End(Not run)
```

---

ds.filter *Keep rows that match a condition*

Description

DataSHIELD implentation of dplyr::filter.

Usage

```
ds.filter(
  df.name = NULL,
  tidy_expr = NULL,
  .by = NULL,
  .preserve = FALSE,
  newobj = NULL,
  datasources = NULL
)
```

Arguments

df.name	Character specifying a serverside data frame or tibble.
tidy_expr	List of expressions that return a logical value, and are defined in terms of the variables in df.name.
.by	Optionally, a selection of columns to group by for just this operation, functioning as an alternative to dplyr::group_by
.preserve	Relevant when the .data input is grouped. If .preserve = FALSE (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping is kept as is.
newobj	Character specifying name for new server-side data frame.
datasources	DataSHIELD connections object.

Value

No return value, called for its side effects. An object (typically a data frame or tibble) with the name specified by newobj is created on the server.

Examples

```
## Not run:
ds.filter(
  df.name = "mtcars",
  tidy_expr = list(cyl == 4 & mpg > 20),
  newobj = "filtered",
  datasources = conns
)

## End(Not run)
```

---

ds.group_by *Group by one or more variables*

Description

DataSHIELD implentation of dplyr::group_by.

Usage

```
ds.group_by(
  df.name = NULL,
  tidy_expr,
  .add = FALSE,
  .drop = TRUE,
  newobj = NULL,
  datasources = NULL
)
```

Arguments

df.name	Character specifying a serverside data frame or tibble.
tidy_expr	List of variables or computations to group by.
.add	When FALSE, the default, group_by() will override existing groups. To add to the existing groups, use .add = TRUE.
.drop	Drop groups formed by factor levels that don't appear in the data? The default is TRUE except when .data has been previously grouped with .drop = FALSE.
newobj	Character specifying name for new server-side data frame.
datasources	DataSHIELD connections object.

Value

No return value, called for its side effects. A grouped data frame with class grouped_df newobj is created on the server, unless the combination of tidy_expr and .add yields a empty set of grouping columns, in which case a tibble will be created on the server.

Examples

```
## Not run:
ds.group_by(
  df.name = "mtcars",
  expr = list(mpg, cyl),
  newobj = "grouped_df"
)

## End(Not run)
```

---

ds.group_keys *Describe the groups within a grouped tibble or data frame*

Description

DataSHIELD implentation of dplyr::group_keys.

Usage

```
ds.group_keys(df.name = NULL, datasources = NULL)
```

Arguments

df.name	Character specifying a serverside tibble.
datasources	DataSHIELD connections object.

Value

A data frame describing the groups.

Examples

```
## Not run:
my_groups <- ds.group_keys("grouped_df")

## End(Not run)
```

`ds.if_else` *Vectorised if-else*

Description

DataSHIELD implementation of `dplyr::if_else`.

Usage

```
ds.if_else(
  condition = NULL,
  true = NULL,
  false = NULL,
  missing = NULL,
  ptype = NULL,
  size = NULL,
  newobj = NULL,
  datasources = NULL
)
```

Arguments

`condition`	A list specifying a logical vector in tidyverse syntax, ie data and column names unquoted.
`true`	Vector to use for TRUE value of condition.
`false`	Vector to use for FALSE value of condition.
`missing`	If not NULL, will be used as the value for NA values of condition. Follows the same size and type rules as true and false.
`ptype`	An optional prototype declaring the desired output type. If supplied, this overrides the common type of true, false, and missing.
`size`	An optional size declaring the desired output size. If supplied, this overrides the size of condition.
`newobj`	Character specifying name for new server-side data frame.
`datasources`	datashield connections object.

Value

No return value, called for its side effects. A vector with the same size as `condition` and the same type as the common type of `true`, `false`, and `missing` is created on the server.

Examples

```
## Not run:
## First log in to a DataSHIELD session with mtcars dataset loaded.


## Refer to the package vignette for more examples.

## End(Not run)
```

`ds.mutate` *Create, modify, and delete columns*

Description

DataSHIELD implementation of `dplyr::mutate`.

Usage

```
ds.mutate(
  df.name = NULL,
  tidy_expr = NULL,
  newobj = NULL,
  .keep = "all",
  .before = NULL,
  .after = NULL,
  datasources = NULL
)
```

Arguments

`df.name`	Character specifying a serverside data frame or tibble.
`tidy_expr`	List of tidyselect syntax to be passed to dplyr::mutate.
`newobj`	Character specifying name for new server-side data frame.
`.keep`	Control which columns from `df.name` are retained in the output. Options include: • `"all"`: Retains all columns from `df.name`. This is the default. • `"used"`: Retains only the columns used in `tidy_expr` to create new columns. • `"unused"`: Retains only the columns not used in `tidy_expr` to create new columns. This is useful if you generate new columns but no longer need the columns used to generate them. • `"none"`: Doesn't retain any extra columns from `df.name`. Only the grouping variables and columns created by `tidy_expr` are kept. Grouping columns and columns created by `tidy_expr` are always kept.
`.before`	<tidy-select> Optionally, control where new columns should appear (the default is to add to the right hand side). See `tidy_expr` for more details.

.after	<tidy-select> Optionally, control where new columns should appear (the default is to add to the right hand side). See tidy_expr for more details.
datasources	datashield connections object.

Value

No return value, called for its side effects. An object (typically a data frame or tibble) with the name specified by newobj is created on the server.

Examples

```
## Not run:
## First log in to a DataSHIELD session with mtcars dataset loaded.

ds.mutate(
  df.name = "mtcars",
  tidy_select = list(mpg_trans = cyl * 1000, new_var = (hp - drat) / qsec),
  newobj = "df_with_new_cols"
)

## Refer to the package vignette for more examples.

## End(Not run)
```

---

ds.rename *Rename columns*

Description

DataSHIELD implentation of dplyr::rename.

Usage

```
ds.rename(df.name = NULL, tidy_expr = NULL, newobj = NULL, datasources = NULL)
```

Arguments

df.name	Character specifying a serverside data frame or tibble.
tidy_expr	List with format new_name = old_name to rename selected variables.
newobj	Character specifying name for new server-side data frame.
datasources	DataSHIELD connections object.

Value

No return value, called for its side effects. An object (typically a data frame or tibble) with the name specified by newobj is created on the server.

Examples

```
## Not run:
## First log in to a DataSHIELD session with mtcars dataset loaded.

ds.rename(
  df.name = "mtcars",
  tidy_select = list(new_var_1 = mpg, new_var_2 = cyl),
  newobj = "df_renamed",
  dataources = conns
)

## Refer to the package vignette for more examples.

## End(Not run)
```

`ds.select` *Keep or drop columns using their names and types*

Description

DataSHIELD implentation of `dplyr::select`.

Usage

```
ds.select(df.name = NULL, tidy_expr = NULL, newobj = NULL, datasources = NULL)
```

Arguments

`df.name`	Character specifying a serverside data frame or tibble.
`tidy_expr`	List of one or more unquoted expressions separated by commas. Variable names can be used as if they were positions in the data frame, so expressions like x:y can be used to select a range of variables.
`newobj`	Character specifying name for new server-side data frame.
`datasources`	DataSHIELD connections object.

Value

No return value, called for its side effects. An object (typically a data frame or tibble) with the name specified by `newobj` is created on the server.

Examples

```
## Not run:
ds.select(
  df.name = "mtcars",
  tidy_expr = list(mpg, starts_with("t")),
  newobj = "df_subset",
  dataources = conns
```

```
)

## End(Not run)
```

`ds.slice` *Subset rows using their positions*

Description

DataSHIELD implentation of `dplyr::slice`.

Usage

```
ds.slice(
  df.name = NULL,
  tidy_expr = NULL,
  .by = NULL,
  .preserve = FALSE,
  newobj = NULL,
  datasources = NULL
)
```

Arguments

`df.name`	Character specifying a serverside data frame or tibble.
`tidy_expr`	List, provide either positive values to keep, or negative values to drop. The values provided must be either all positive or all negative. Indices beyond the number of rows in the input are silently ignored.
`.by`	Optionally, a selection of columns to group by for just this operation, functioning as an alternative to `dplyr::group_by`
`.preserve`	Relevant when the .data input is grouped. If .preserve = FALSE (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping is kept as is.
`newobj`	Character specifying name for new server-side data frame.
`datasources`	DataSHIELD connections object.

Value

No return value, called for its side effects. An object (typically a data frame or tibble) with the name specified by `newobj` is created on the server.

Examples

```
## Not run:
ds.slice(
  df.name = "mtcars",
  expr = list(1:10),
  .by = "cyl",
  newobj = "sliced_df"
)

## End(Not run)
```

`ds.ungroup` *Remove grouping from a tibble or data frame*

Description

DataSHIELD implentation of `dplyr::ungroup`.

Usage

```
ds.ungroup(x = NULL, newobj = NULL, datasources = NULL)
```

Arguments

`x`	a tibble or data frame.
`newobj`	Character specifying name for new server-side data frame.
`datasources`	DataSHIELD connections object.

Value

No return value, called for its side effects. An ungrouped data frame or tibble is created on the server.

Examples

```
## Not run:
ds.ungroup("grouped_df")

## End(Not run)
```

Index