Package 'spdplyr'

May 12, 2020

Type Package

Version 0.4.0

Title Data Manipulation Verbs for the Spatial Classes

Description Methods for 'dplyr' verbs for 'sp' 'Spatial' classes. The basic verbs that modify data attributes, remove or re-arrange rows are supported and provide complete 'Spatial' analogues of the input data. The group by and summarize work flow returns a non-topological spatial union. There is limited support for joins, with left and inner to copy attributes from another table.

URL https://github.com/mdsumner/spdplyr

BugReports https://github.com/mdsumner/spdplyr/issues

Depends R (>= 3.2.3), dplyr, sp

Imports methods, rlang, spbabel, tibble, utils

Suggests testthat, maptools, raster, rmarkdown, knitr, covr, spelling

VignetteBuilder knitr

LazyData yes

License GPL-3

RoxygenNote 7.1.0

Language en-US

NeedsCompilation no

Author Michael D. Sumner [aut, cre],
Joscha Legewie [ctb],
Jussi Jousimo [ctb]

Maintainer Michael D. Sumner <mdsumner@gmail.com>

Repository CRAN

Date/Publication 2020-05-12 11:40:02 UTC

R topics documented:

`spdplyr-package` *dplyr for sp*

Description

Data Manipulation Verbs for Spatial Classes.

Details

The spdplyr package provides methods for the dplyr verbs.

O. dplyr verbs

`filter`	filter
`slice`	slice
`arrange`	arrange
`select`	select
`rename`	rename
`distinct`	distinct
`mutate`	mutate
`transmute`	transmute
`group_by`	group_by
`summarise`	summarise (or summarize)

I. dplyr-Spatial

dplyr verbs operating directly on Spatial objects The implementation of `group_by` requires that sp objects are able use the object extending class "data.frame", `summarise` acts on the attributes in the table according to the expressions use and on the geometry by performing a non-topological union.

dplyr-S3 *dplyr S3 methods*

Description

Worker functions used by dplyr features.

Usage

```
## S3 method for class 'Spatial'
tbl_vars(x)

## S3 method for class 'Spatial'
groups(x)
```

Arguments

x	input Spatial object

Details

These will work for the Spatial-DataFrame objects, though not properly for any-Spatial.

dplyr-Spatial *Dplyr verbs for Spatial*

Description

Direct application of the dplyr verbs to Spatial objects. There is no need for a conversion from and to Spatial with this approach. Not all verbs are supported, see Details.

Usage

```
## S3 method for class 'Spatial'
mutate(.data, ...)

## S3 method for class 'Spatial'
transmute(.data, ...)

## S3 method for class 'Spatial'
summarise(.data, ...)

## S3 method for class 'Spatial'
group_by(.data, ...)

## S3 method for class 'Spatial'
```

```
filter(.data, ...)

## S3 method for class 'Spatial'
arrange(.data, ...)

## S3 method for class 'Spatial'
slice(.data, ...)

## S3 method for class 'Spatial'
select(.data, ...)

## S3 method for class 'Spatial'
rename(.data, ...)

## S3 method for class 'Spatial'
distinct(.data, ..., .keep_all = FALSE)

## S3 method for class 'Spatial'
left_join(x, y, by = NULL, copy = FALSE, ...)

## S3 method for class 'Spatial'
inner_join(x, y, by = NULL, copy = FALSE, ...)
```

Arguments

.data	A tbl.
...	Name-value pairs of expressions. See mutate_
.keep_all	argument for distinct, we have to set it to TRUE
x	A pair of data frames, data frame extensions (e.g. a tibble), or lazy data frames (e.g. from dbplyr or dtplyr). See *Methods*, below, for more details.
y	tbl to join
by	A character vector of variables to join by. If NULL, the default, *_join() will perform a natural join, using all variables in common across x and y. A message lists the variables so that you can check they're correct; suppress the message by supplying by explicitly. To join by different variables on x and y, use a named vector. For example, by = c("a" = "b") will match x$a to y$b. To join by multiple variables, use a vector with length > 1. For example, by = c("a","b") will match x$a to y$a and x$b to y$b. Use a named vector to match different variables in x and y. For example, by = c("a" = "b","c" = "d") will match x$a to y$b and x$c to y$d. To perform a cross-join, generating all combinations of x and y, use by = character().
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.

Details

mutate, transmute, filter, arrange, slice, select, rename, distinct all work with attributes on the "data" slot and leave the geometry unchanged.

summarise collapses to a grouped geometries by listing all subgeometries together, it does not perform any topological union or merge, and it takes no account of the calculations done on attributes. This is a brutal collapse of all the data, and is identical to what is seen with spplot(x, "group"). The behaviour of geometric collapse like this is touch and go anyway, see the examples for a what ‘rgeos::gUnion‘ does.

summarise for points and multipoints, ... todo single Multipoint for multiple points

Warning

‘distinct‘ uses behaviour identical to ‘duplicated‘, by coercing all the relevant values to text and determining uniqueness from those. ‘dplyr::distinct‘ uses a different internal method that will give different results for some cases of numeric data.

Note

Beware that attributes stored on Spatial objects *are not* linked to the geometry. Attributes are often used to store the area or perimeter length or centroid values but these may be completely unmatched to the underlying geometries.

Examples

```
library(sp)
library(maptools)
data(wrld_simpl)
library(spdplyr)
library(raster)
wrld_simpl %>% mutate(NAME = "allthesame", REGION = row_number())
wrld_simpl %>% transmute(alpha = paste0(FIPS, NAME))
wrld_simpl %>% filter(NAME %in% c("New Zealand", "Australia", "Fiji"))
## Not run:
wrld_simpl %>% arrange(LON)
wrld_simpl %>% slice(c(9, 100))
wrld_simpl %>% dplyr::select(UN, FIPS)
wrld_simpl %>% rename(`TM_WORLD_BORDERS_SIMPL0.2NAME` = NAME)
wrld_simpl %>% distinct(REGION, .keep_all = TRUE) %>%
   arrange(REGION)  ## first alphabetically in REGION
wrld_simpl %>% arrange(REGION, desc(NAME)) %>% distinct(REGION, .keep_all = TRUE) ## last

## End(Not run)
## we don't need to use piping
slice(filter(mutate(wrld_simpl, likepiping = FALSE), abs(LON - 5) < 35 & LAT > 50), 4)


## works with Lines
#as(wrld_simpl, "SpatialLinesDataFrame") %>%
 # mutate(perim = rgeos::gLength(wrld_simpl, byid = TRUE))
```

```
## Not run:
## summarise/ze can be used after group_by, or without
wrld_simpl %>% filter(REGION == 150) %>% summarize(max(AREA))
wrld_simpl %>% group_by(REGION) %>% summarize(max(AREA)) %>%
plot(col = rainbow(nlevels(factor(wrld_simpl$REGION)), alpha = 0.3))

## End(Not run)
## group_by and summarize

## Not run:
g <- wrld_simpl  %>% group_by(REGION)  %>%
 summarize(alon = mean(LON), mxlat = max(LAT), mxarea = max(AREA))
g %>% mutate(ar = factor(REGION)) %>% spplot("ar")
w <- wrld_simpl
w$ar <- factor(w$REGION)
spplot(w, "ar")

## End(Not run)
## Not run:
# compare what rgeos gives
##spplot(rgeos::gUnionCascaded(w, id = w$ar))  ## good grief, is this compelling . . .
## this is hardly a clean dissolve
##plot(rgeos::gUnionCascaded(w, id = w$ar), col = rainbow(nlevels(factor(w$ar)), alpha = 0.5))

## End(Not run)
```

`mpoint1` *MultiPointsDataFrame data set*

Description

MultiPointsDataFrame data set

`show,Spatial-method` *sp methods*

Description

Sp methods

Usage

```
## S4 method for signature 'Spatial'
show(object)

## S4 method for signature 'SpatialPoints'
show(object)
```

Arguments

object	Spatial object

---

spmap *"South-east" map data.*

Description

Created in /data-raw/ spmap is a subset of wrld_simpl from maptools.

Examples

```
library(dplyr)
spmap %>% filter(NAME == "Antarctica")
```

Index