Package ‘csquares’

July 25, 2024

Title Concise Spatial Query and Representation System (c-Squares)

Version 0.0.7

Author Pepijn de Vries [aut, cre] (<https://orcid.org/0000-0002-7961-6646>)

Description Encode and decode c-squares, from and to simple feature (sf) or spatiotemporal arrays (stars) objects. Use c-squares codes to quickly join or query spatial data.

Imports dplyr, methods, purrr, rlang, sf, stars, stringr, tidyr, tidyselect, vctrs

Suggests curl, DiagrammeR, ggplot2, knitr, lifecycle, rmarkdown, maturaleza, maturaleza2data, testthat (>= 3.0.0)

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.2.3

Depends R (>= 4.1.0)

LazyData true


Config/testthat/edition 3


BugReports https://github.com/pepijn-devries/csquares/issues

VignetteBuilder knitr

NeedsCompilation no

Maintainer Pepijn de Vries <pepijn.devries@outlook.com>

Repository CRAN

Date/Publication 2024-07-25 12:00:06 UTC
Contents

as_csquares ................................................................. 2
drop_csquares ............................................................. 4
expand_wildcards ......................................................... 5
format.csquares ............................................................ 6
ices_centroids .............................................................. 7
ices_columns ................................................................. 9
in_csquares ................................................................. 9
join ................................................................. 10
new_csquares ............................................................... 12
orca ............................................................. 13
resample_csquares ......................................................... 14
st_as_sf ............................................................... 15
st_as_stars.csquares ....................................................... 16
tidyverse ................................................................. 17
validate_csquares .......................................................... 20
vctrs ............................................................. 21

Index 23

as_csquares  

Convert lon-lat coordinates into c-square codes

Description

Takes WGS84 longitude and latitude coordinates and finds the closest matching c-squares for a given resolution.

Usage

as_csquares(x, resolution, csquares, ...)

## Default S3 method:
as_csquares(x, resolution, csquares, ...)

## S3 method for class 'character'
as_csquares(x, resolution, csquares, validate = TRUE, ...)

## S3 method for class 'numeric'
as_csquares(x, resolution = 1, csquares, ...)

## S3 method for class 'data.frame'
as_csquares(x, resolution = 1, csquares, ...)

## S3 method for class 'sf'
as_csquares(x, resolution = 1, csquares, ..., use_centroids = TRUE)
## S3 method for class 'sfc'
as_csquares(x, resolution = 1, csquares, ..., use_centroids = TRUE)

## S3 method for class 'stars'
as_csquares(x, resolution = 1, csquares, ...)

Arguments

- **x**: An object to be coerced to a csquares object. `x` can be a vector of character strings representing c-squares code. It can also be a numeric matrix with two columns containing the x and y coordinates. `x` can also be a simple features object (sf) or a spatial arrays object (stars).

- **resolution**: Resolution (in WGS84 degrees) to be used for creating c-squares codes. As per c-square specifications, the resolution should be 10 or less, yet greater than 0. It should be a tenfold of 1 or 5. Valid resolutions are therefore: 10, 5, 1, 0.5, 0.1, etc.

- **csquares**: If `x` is not a vector of character strings (but for instance a data.frame), the `csquares` argument should specify the name of the element of `x` containing the c-square codes as character strings.

- **...**: Currently ignored

- **validate**: A logical value indicating whether the created object needs to be validated. Defaults to TRUE. Validation can be time-consuming so set to FALSE to save computing time.

- **use_centroids**: In case `x` is a simple features object and `use_centroids` is TRUE, the centroid of each geometry is used for deriving c-squares. If it is FALSE all coordinates in the geometry are used.

Value

Returns a csquares object that contains c-squares codes.

Author(s)

Pepijn de Vries

Examples

```r
as_csquares(cbind(x = 5.2399066, y = 52.7155812), resolution = 1)
onca_csq <- as_csquares(orca, csquares = "csquares")
```
Description

Drops c-square data from an object, but keeps the parent class of the object intact. You cannot
deselect the csquare column from a csquares object as this will render the object invalid. Use
drop_csquares instead.

Usage

drop_csquares(x, ...)

Arguments

x  
An object of class csquares from which the c-square information needs to be
dropped.

...  
ignored

Value

Returns a copy of x inheriting its parent classes but with out csquares info.

Author(s)

Pepijn de Vries

Examples

```r
csq <- as_csquares("1000")
drop_csquares(csq)

csq <-
data.frame(csquares = "1000", foo = "bar") |>  
as_csquares(csquares = "csquares")

drop_csquares(csq)
```
**expand_wildcards**  
Expand c-squares with wildcards to all matching c-squares

**Description**

The asterisk (*) can be used as a wildcard, for a compact notation of csquares. expand_wildcards will replace all wild cards with valid combinations of values and expands the compact notation to an explicit notation without wildcards. Check out vignette("wildcards") for more details.

**Usage**

`expand_wildcards(x, csquares, ...)`

**Arguments**

- **x**  
  A character string containing csquares codes with wildcards (asterisk character); or a data.frame that contains a column with csquares codes with wildcards

- **csquares**  
  When `x` is data.frame this argument should specify the column name that contains the csquares codes with wildcards.

- **...**  
  ignored

**Value**

Returns a csquares object with full notation

**Author(s)**

Pepijn de Vries

**Examples**

```r
expand_wildcards("1000:*")
expand_wildcards("1000:***")
expand_wildcards("1000:***")
expand_wildcards("1000:***:")
expand_wildcards(c("1000:*", "1000:***", "1000:***", "1000:***:*"))
expand_wildcards(data.frame(csq = "1000:*", foo = "bar"), csquares = "csq")
```
Description

Basic S3 methods for handling csquares objects

Usage

```r
## S3 method for class 'csquares'
format(x, ...)  

## S3 method for class 'csquares'
print(x, short = TRUE, ...)  

## S3 method for class 'csquares'
as.character(x, ...)  

## S3 method for class 'csquares'
summary(object, ...)  

## S3 method for class 'csquares'
as.data.frame(x, ...)  

data.frame.csquares(...)  

c(...)  

## S3 method for class 'csquares'
rbind(..., deparse.level = 1)  

## S3 method for class 'csquares'
cbind(..., deparse.level = 1)  

## S3 method for class 'csquares'
x[i, j, ..., drop = FALSE]  

## S3 method for class 'csquares'
x[[i]]  

## S3 method for class 'csquares'
x$name  

## S3 replacement method for class 'csquares'
x[i, j] <- value
```
## S3 method for class 'csquares'
merge(x, y, ...)

## S3 replacement method for class 'csquares'
names(x) <- value

### Arguments

- **x**: A csquares object to be handled by the s3 methods
- **object**: A csquares object to be handled by the s3 methods
- **...**: Passed on to generic methods
- **short**: logical option to print csquares vctrs_vec. If TRUE it will only print one line, if FALSE it will print up to options("max.print") records.
- **deparse.level**: integer controlling the construction of labels in the case of non-matrix-like arguments (for the default method):
  - deparse.level = 0 constructs no labels;
  - the default deparse.level = 1 typically and deparse.level = 2 always construct labels from the argument names, see the ‘Value’ section below.
- **i, j, name**: Indices/name for selecting subsets of x
- **drop**: logical value indicating if unused dimensions should be dropped
- **value**: Replacement values for a subset. A csquares object or a character string that can be coerced to a csquares object
- **y**: A data.frame to be merged with x

### Value

Returns (a subsetted / formatted / modified version of) x

---

**ices_centroids** Get ICES geometries

**Description**

[Experimental] Functions to convert ICES rectangles
Usage

ices_centroids(ices_rect)
ices_rectangles(ices_rect)
ices_to_csquares(ices_rect)
ices_from_csquares(csquares)

Arguments

ices_rect  A character vector containing valid ICES rectangle codes
csquares   A csquares object, or an object that can be coerced with as_csquares().

Value

In case of ices_centroids a sf::st_sf() object is returned, with POINT geometries representing
the centroids of the ICES rectangles.

In case of ices_rectangles a sf::st_sf() object is returned, with POLYGON geometries representing
the outline of the ICES rectangles.

In case of ices_to_csquares a csquares object inheriting from sf::st_sf() is returned, the
csquares code should represent the ICES rectangles.

In case of ices_from_csquares a character vector is returned with ICES rectangle codes that
 correspond with the csquares. The method is fast yet crude: it only checks in which ICES rectangles
the centroids of the csquares are located. It does not check if the resolution matches. NA values are
 returned when csquares are situated outside the area covered by ICES rectangles.

Author(s)

Pepijn de Vries

Examples

ices_rects <-
   "32F2", "33F2", "34F2", "35F2",
   "31F3", "32F3", "33F3", "34F3", "35F3",
   "31F4", "32F4", "33F4", "34F4", "35F4")
ices_centroids(ices_rects)
ices_rectangles(ices_rects)
ices_csq <- ices_to_csquares(ices_rects)
ices_from_csquares(ices_csq)
**ices_columns**

| ices_columns | Valid ICES rectangle columns |

**Description**

[Experimental] Get all valid column codes of ICES rectangles. Note that ICES subrectangles are not compatible with csquares. For more details see vignette("ices").

**Usage**

ices_columns()

**Value**

A character vector with all allowed codes for the columns in ICES rectangles.

**Examples**

ices_columns()

---

**in_csquares**

Match c-squares against other c-squares (with wildcards)

**Description**

Checks if csquares codes in table matches values in x. Wildcards are allowed in table for this comparison. Check out vignette("wildcards") for more details.

**Usage**

in_csquares(x, table, strict = FALSE, mode = "any", ...)

**Arguments**

- **x**
  - An object of class 'csquares' that will be checked for matching values in table
- **table**
  - A character string representing a csquares code. The code can contain wildcards (asterisk * and percentage % characters, both having identical meaning). Any symbol in x will result in a positive match against the wildcard. table can also be of class csquares, but these objects cannot contain wildcards.
- **strict**
  - When set to FALSE, a match is positive when the start of x, matches against values in table, even when x has a higher resolution. When set to TRUE, a match is only positive when the resolution of x and table is identical.
- **mode**
  - Two modes are allowed: "all" and "any". When an element of x consists of multiple raster cells, it the mode will determine whether a match is positive or not. In case of "all", all raster cells in the element of x need to match with the cells in table, for a positive match. In case of "any", any match will do.
- **...**
  - Ignored
**Value**

Returns a vector of logical values with the same number of elements or rows as x

**Author(s)**

Pepijn de Vries

**Examples**

```r
library(dplyr)

in_csquares(orca$csquares, c("3400:2", "5515:3"))
in_csquares(orca$csquares, "3400:2|5515:3")

## Percentage symbols are interpreted the same as asterisk symbols
## both are wild cards
in_csquares(orca$csquares, "1%%%:%") |> table()

## Same as above
in_csquares(orca$csquares, "1***:*") |> table()

## Also same as above
in_csquares(orca$csquares, "1***", strict = FALSE) |> table()

## Strict interpretation results in no matches
in_csquares(orca$csquares, "1***", strict = TRUE) |> table()

## Filter orca data to North Eastern quadrant (1***:* only:
orca |> filter(in_csquares(csquares, "1***:*")) |> nrow()
```

---

**join**

*Join csquares objects using tidyverse conventions*

**Description**

When a csquares object inherits from class `data.frame`, you can apply tidyverse joins to the object (?dplyr::join). The functions implemented here make sure that the csquares properties are preserved. The functions should be called via the `dplyr` generics. So load the `dplyr` package first, then call the function without the .csquares suffix (see examples). When x inherits from `stars`, only `left_join` is supported.
join

Usage

inner_join.csquares(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)

left_join.csquares(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)

right_join.csquares(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)

full_join.csquares(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)

semi_join.csquares(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)

anti_join.csquares(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)

st_join.csquares(x, y, join, ..., suffix)

Arguments

x, y  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.

by  A join specification created with join_by(), or 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 on different variables between x and y, use a join_by() specification. For example, join_by(a == b) will match x$a to y$b.

To join by multiple variables, use a join_by() specification with multiple expressions. For example, join_by(a == b, c == d) will match x$a to y$b and x$c to y$d. If the column names are the same between x and y, you can shorten this by listing only the variable names, like join_by(a, c).

join_by() can also be used to perform inequality, rolling, and overlap joins. See the documentation at ?join_by for details on these types of joins.

For simple equality joins, you can alternatively specify a character vector of variable names to join by. For example, by = c("a", "b") joins x$a to y$a and x$b to y$b. If variable names differ between x and y, use a named character vector like by = c("x_a" = "y_a", "x_b" = "y_b").

To perform a cross-join, generating all combinations of x and y, see cross_join().

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.

suffix  If there are non-joined duplicate variables in x and y, these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2.

...  Other parameters passed onto methods.

join  geometry predicate function with the same profile as st_intersects; see details
new_csquares

Create a c-squares raster from a bounding box

Description

Creates a spatial raster (stars) with c-square codes for a specified bounding box, using a specified resolution. The raster will be conform c-squares specifications.

Usage

new_csquares(x, resolution = 1, crs = 4326)

Arguments

x       An object of class bbox or an object that can be coerced to a bbox. It defines the bounding box for the c-squares grid created by this function.
resolution  Resolution (in WGS84 degrees) to be used for creating c-squares codes. As per c-square specifications, the resolution should be 10 or less, yet greater than 0. It should be a tenfold of 1 or 5. Valid resolutions are therefore: 10, 5, 1, 0.5, 0.1, etc.
crs        The projection to be used for the created grid. By default it is WGS84 (EPSG:4326).

Value

Returns a stars and csquares object based on the provided bounding box and resolution.

Examples

```r
if (requireNamespace(c("sf", "dplyr"))) {
  library(csquares)
  library(sf)
  library(dplyr)
  orca_sf <- orca |> as_csquares(csquares = "csquares") |>
    st_as_sf()
  right_table <- data.frame(csquares = c("1000:1", "1004:1"), foo = "bar")
  orca_join <- left_join(orca_sf, right_table, by = "csquares")
  orca_join <- right_join(orca_sf, right_table, by = "csquares")
  orca_join <- inner_join(orca_sf, right_table, by = "csquares")
  orca_join <- anti_join(orca_sf, right_table, by = "csquares")
  orca_join <- semi_join(orca_sf, right_table, by = "csquares")
  orca_grid <- new_csquares(orca_sf, 5)
  orca_grid <- left_join(orca_grid, orca, by = "csquares")
}
```
**Author(s)**

Pepijn de Vries

**Examples**

```r
library(sf)
nc <- st_read(system.file("shape/nc.shp", package = "sf"))
new_csquares(nc)
```

<table>
<thead>
<tr>
<th>orca</th>
<th>Killer whale realm</th>
</tr>
</thead>
</table>

**Description**

Killer whale realm

**Usage**

orca

**Format**

orca:
The orca object is a Killer whale realm data set extracted from the data as provided by Costello (2017) and published by Costello et al. (2017). It is a data frame with 2,058 rows and two columns:

- **csquares** c-squares codes indicating spatial grid cells
- **orcinus_orca** logical values indicating whether the corresponding c-squares grid cell belongs to the killer whales (Orcinus orca) biogeographic realm or not.

**References**

- Costello, M.J. (2017); University of Auckland doi:10.17608/k6.auckland.5086654 Licence CC BY 4.0
- Costello M.J., Tsai P., Wong P.S., Cheung A.K.L, Basher Z. & Chaudhary C. (2017); "Marine biogeographic realms and species endemicity" Nature Communications 8, 1057 doi:10.1038/s41467017011212
### resample_csquares

Resample csquares to a different resolution

#### Description

Resample csquares objects to higher or lower resolutions.

#### Usage

```r
resample_csquares(x, method = "target", ..., resolution, magnitude = 1L)
```

#### Arguments

- **x**: A csquares object to be resampled to a different resolution
- **method**: Method for determining the resolution of the resulting csquares. Should be one of "target", "min", "max", "up", or "down". "target" will resample x to the level specified with resolution.
- **...**: When x inherits the stars class and the resulting object has a lower resolution than x, the dots are passed on to dplyr::summarise(). This allows you to summarise columns to the lower resolution.
- **resolution**: Resolution (in WGS84 degrees) to be used for creating c-squares codes. As per c-square specifications, the resolution should be 10 or less, yet greater than 0. It should be a tenfold of 1 or 5. Valid resolutions are therefore: 10, 5, 1, 0.5, 0.1, etc.
- **magnitude**: When method == "up" or "down", this parameter specifies the number of steps to increase or decrease the resolution. Should be a positive integer.

#### Value

A csquares object based on x

#### Author(s)

Pepijn de Vries

#### Examples

```r
csq <- as_csquares(c("1000", "5000:2|5000:100", "3000:100:100"))
csq_df <- as_csquares(data.frame(csq = csq, foobar = letters[1:3]), csquares = "csq")

## Resample csquares based on the one with the lowest resolution:
resample_csquares(csq, "min")

## Resample csquares to a specific resolution
resample_csquares(csq, "target", resolution = 5)

## Same, but applied to a csquares object inheriting from a data.frame
```
st_as_sf

Create a simple features object from c-squares

Description

Converts a character string of c-squares in a spatially explicit simple features object (sf). It can also convert data.frames with a column of c-squares codes to an sf object.

Usage

```
st_as_sf.csquares(x, ..., use_geometry = TRUE)

st_as_sfc.csquares(x, ..., use_geometry = TRUE)
```
### st_as_stars.csquares

**Coerce csquares object into a stars object**

Take a csquares object created with new_csquares or as_csquares and coerce it to a spatiotemporal array (stars).

**Usage**

```r
st_as_stars.csquares(x, ...)
```

**Arguments**

- **x**: An object of class csquares created with new_csquares or as_csquares
- **...**: ignored.

**Value**

Returns a spatiotemporal array (stars) object based on x.
**tidyverse**

**Author(s)**

Pepijn de Vries

**Examples**

```r
library(stars)
st_as_stars(as_csquares("7500:110:3|7500:110:1|1500:110:3|1500:110:1"))
st_as_stars(as_csquares(orca, csquares = "csquares"))
```

---

**Description**

Tidyverse methods for csquares objects that inherit from `data.frame`, `tibble`, `sf`, or in some cases `stars`. Load the tidyverse package containing the generic implementation (`dplyr` or `tidyr`), and call the function without the `.csquares` suffix. See examples and `vignette("tidy")` for more details. The methods implemented here ensure that the `csquare` class is preserved.

**Usage**

```r
filter.csquares(.data, ..., .dots)
select.csquares(.data, ...)
as_tibble.csquares(x, ...)
arrange.csquares(.data, ..., .dots)
group_by.csquares(.data, ..., add = FALSE)
ungroup.csquares(.data, ...)
rowwise.csquares(.data, ...)
mutate.csquares(.data, ..., .dots)
rename.csquares(.data, ...)
rename_with.csquares(.data, .fn, .cols, ...)
slice.csquares(.data, ..., .dots)
distinct.csquares(.data, ..., .keep_all = FALSE)
summarise.csquares(.data, ..., .dots)
```
pivot_longer.csquares(
  data,
  cols,
  ..., 
  cols_vary = "fastest",
  names_to = "name",
  names_prefix = NULL,
  names_sep = NULL,
  names_pattern = NULL,
  names_ptypes = NULL,
  names_transform = NULL,
  names_repair = "check_unique",
  values_to = "value",
  values_drop_na = FALSE,
  values_ptype = NULL,
  values_transform = NULL
)

pivot_wider.csquares(
  data,
  ..., 
  id_cols = NULL,
  id_expand = FALSE,
  names_from = NULL,
  names_prefix = ",",
  names_sep = ",",
  names_glue = NULL,
  names_sort = FALSE,
  names_vary = "fastest",
  names_expand = FALSE,
  names_repair = "check_unique",
  values_from = NULL,
  values_fill = NULL,
  values_fn = NULL,
  unused_fn = NULL
)

group_split.csquares(.tbl, ..., .keep = TRUE)
nest.csquares(.data, ...)

unite.csquares(data, col, ..., sep = ",", remove = TRUE)
unnest.csquares(data, ..., .preserve = NULL)
unnest.csquares_nested(data, cols, ...)
drop_na.csquares(x, ...)
Arguments

Passed to tidyverse generic methods. Consult their documentation.

Details

Note that the implementation of summarise.csquares has changed since version 0.0.5.002, to better reflect the dplyr generic implementation. To get results similar to the earlier implementation please use resample_csquares().

Author(s)

Pepijn de Vries

Examples

```r
if (requireNamespace(c("dplyr", "tidyr"))) {
  library(dplyr)
  library(tidyr)

  ## Create a csquares object from the orca dataset:
  orca_csq <- as_csquares(orca, csquares = "csquares")

  ## Filter values that belong to the killer whale realm:
  orca2 <- filter(orca_csq, orcinus_orca == TRUE)

  ## Mutate the object to hold information on the quadrant:
  orca_csq <- mutate(orca_csq, quadrant = csquares |> as.character() |> substr(1,1))

  ## Select the quadrant column:
  orca2 <- select(orca_csq, quadrant)

  ## Convert it into a tibble:
  orca_csq <- as_tibble(orca_csq)

  ## Arrange by quadrant:
  orca2 <- arrange(orca_csq, quadrant)

  ## Group by quadrant:
  orca_csq <- group_by(orca_csq, quadrant)

  ## Summarise per quadrant:
  summarise(orca_csq, realm_frac = sum(orcinus_orca)/n())

  #' Introduce a group split:
```
## Ungroup the object:
\>
## Take a slice of the first three rows:
\>
## Take a sample of 10 rows with replacement:
\>
## Rename a column:
\>
## Distinct will remove any duplicated rows:
\>
## Pivot to a wide format:
\>
## Pivot to a long format (note that you can’t pivot the csquares column to long)
\>
## Unite two columns into one:
\>
## As the csquares column gets nested in the example below,
## the resulting object is no longer of class csquares:
\>
## Unnest it:
\>
### validate_csquares

#### Test if a csquares object is valid

**Description**
Tests if a csquares object is correctly specified and can be translated into valid coordinates

**Usage**
validate_csquares(x)
**Arguments**

- **x**: An object of class `csquares` to be evaluated.

**Value**

Returns a logical value indicating whether the `csquares` object is valid or not.

**Author(s)**

Pepijn de Vries

**Examples**

```r
validate_csquares(
  as_csquares("7500:110:3|7500:110:1|1500:110:3|1500:110:1")
)
```

---

**Description**

Implementations to support csquare vctrs operations. There is no need to call these functions directly.

**Usage**

```r
vec_cast.csquares(x, to, ...)
```

---

```r
vec_cast.csquares(x, to, ...)
## S3 method for class 'csquares'
vec_cast.csquares(x, to, ...)

vec_cast.csquares(x, to, ...)
## S3 method for class 'character'
vec_cast.csquares(x, to, ...)

vec_cast.csquares(x, to, ...)
## Default S3 method:
vec_cast.csquares(x, to, ...)

vec_ptype2.csquares(x, y, ...)
```

---

```r
vec_ptype2.csquares(x, y, ...)
## S3 method for class 'character'
vec_ptype2.csquares(x, y, ...)

vec_ptype2.csquares(x, y, ...)
## S3 method for class 'csquares'
vec_ptype2.csquares(x, y, ...)

vec_ptype2.csquares(x, y, ..., x_arg = "x", y_arg = "y")
```
Arguments

x, y              Vector types.
to                Types to cast to. If NULL, x will be returned as is.
...               Ignored.
x_arg, y_arg      Argument names for x and y.
Index

* datasets
  orca, 13
  ?join_by, 11
  [.csquares (format.csquares), 6
  [<-.csquares (format.csquares), 6
  [[-.csquares (format.csquares), 6
  $ .csquares (format.csquares), 6
  $<-.csquares (format.csquares), 6
  anti_join.csquares (join), 10
  arrange.csquares (tidyverse), 17
  as.character.csquares
    (format.csquares), 6
  as.data.frame.csquares
    (format.csquares), 6
  as.csquares, 2, 16
  as_tibble.csquares (tidyverse), 17
  bbox, 12
  c.csquares (format.csquares), 6
  cbind.csquares (format.csquares), 6
  cross_join(), 11
  data.frame.csquares (format.csquares), 6
  distinct.csquares (tidyverse), 17
  drop_csquares, 4
  drop_na.csquares (tidyverse), 17
  expand_wildcards, 5
  filter.csquares (tidyverse), 17
  format.csquares, 6
  full_join.csquares (join), 10
  group_by.csquares (tidyverse), 17
  group_split.csquares (tidyverse), 17
  ices_centroids, 7
  ices_columns, 9
  ices_from_csquares (ices_centroids), 7
  ices_rectangles (ices_centroids), 7
  ices_to_csquares (ices_centroids), 7
  in_csquares, 9
  inner_join.csquares (join), 10
  join, 10
  join_by(), 11
  left_join.csquares (join), 10
  merge.csquares (format.csquares), 6
  mutate.csquares (tidyverse), 17
  names<- .csquares (format.csquares), 6
  nest.csquares (tidyverse), 17
  new_csquares, 12, 16
  orca, 13
  pivot_longer.csquares (tidyverse), 17
  pivot_wider.csquares (tidyverse), 17
  print.csquares (format.csquares), 6
  rbind.csquares (format.csquares), 6
  rename.csquares (tidyverse), 17
  rename_with.csquares (tidyverse), 17
  resample_csquares, 14
  right_join.csquares (join), 10
  rowwise.csquares (tidyverse), 17
  select.csquares (tidyverse), 17
  semi_join.csquares (join), 10
  sf, 3, 15, 16
  sfc, 16
  slice.csquares (tidyverse), 17
  st_as_sf, 15
  st_as_sfc (st_as_sf), 15
  st_as_stars.csquares, 16
  st_intersects, 11
  st_join.csquares (join), 10

23
stars, 3, 12, 16
summarise.csquares (tidyverse), 17
summary.csquares (format.csquares), 6
tidyverse, 17
ungroup.csquares (tidyverse), 17
unite.csquares (tidyverse), 17
unnest.csquares (tidyverse), 17
unnest.csquares_nested (tidyverse), 17
validate_csquares, 20
vctrs, 21
vec_cast.csquares (vctrs), 21
vec_ptype2.csquares (vctrs), 21