fars.R

carlos

Sat Sep 2 21:05:51 2017

Read file into a dplyr's data frame tbl

This function reads a file by firstly checking whether a files with that name exists and then converting in into a data frame table

@param filename A character string giving the file's name the function will read

@return This function returns either a error message if the file's name does not exits or a data frame table representing the data of the file

@examples fars_read("./data/accident_2015.csv.bz2") fars_read("mistake_file_name")

@import readr dplyr @export

```
fars_read <- function(filename) {
        if(!file.exists(filename))
                stop("file '", filename, "' does not exist")
        data <- suppressMessages({
                readr::read_csv(filename, progress = FALSE)
        })
        dplyr::tbl_df(data)
}
```

Make a valid filename given a year

This function takes a year in the format yyyy (e.g 1977, 2007, etc) and it gives back a valid fars filename.

@param year a string or integer of four digits indicating the wanted year

@return as a side effect, this functions prints a string with a valid fars filename such as 'accident_1997.csv.bz2'

@examples make_filename(2001) make_filname(1987)

@export

```
make_filename <- function(year) {
        year <- as.integer(year)
        sprintf("accident_%d.csv.bz2", year)
}



## magrittr needed for the operator %>%
```

Reads years and returns list with corresponding months and years

This function reads a vector or list 'years' with four digit number in the format yyyy (e.g., 1927) and if the years exist/are correct the function wil returns a list with the correspondung month and year found in the data

@param years It can be a vector or a list of number representing years with format 'yyyy', such as in '1927'

@return If 'years' has element corresponding with data bases's years a list of months corresponding to the/these year/s is given back other wise a exception error is thrown.

@examples fars_read_years(years = c(2013, 2015)) fars_read_years(years = 2013:2015) fars_read_years(years = list(2013,2015))

@export

```
fars_read_years <- function(years) {
        lapply(years, function(year) {
                file <- make_filename(year)
                #print(file)
                tryCatch({
                        dat <- fars_read(file)
                        #head(dat)
                        dplyr::mutate(dat, year = year) %>%
                                dplyr::select(MONTH, year)
                }, error = function(e) {
                        warning("invalid year: ", year)
                        return(NULL)
                })
        })
}
```

Takes years in format yyyy and returns the number of observations per month per year

@inheritParams fars_read_years

@return a dplyr tibble object with the number of observations per month per year if the later are found to be describe in the data objects. Months are displayed as a column and years as colnames; number of observations are printed under years corresponding by months

@examples fars_summarize_years(years = c(2013, 2015)) fars_summarize_years(years = 2013:2015) fars_summarize_years(years = list(2013,2015))

@import tidyr magrittr @export

```
fars_summarize_years <- function(years) {
        dat_list <- fars_read_years(years)
        dplyr::bind_rows(dat_list) %>%
                dplyr::group_by(year, MONTH) %>%
                dplyr::summarize(n = n()) %>%
                tidyr::spread(year, n)
}
```

Plot a map with geolocated observations (cars accidents)

This functions takes a valid state number and a valid year and produces a map with geolocated observations (accidents) plotted as dots on the map.

@param state.num valid State number (between 1 and 51) @inheritParams fars_read_years

@return A map with geolocated accidents as dots on the map

@examples fars_map_state(51, 2015)

@import maps graphics @export

```
fars_map_state <- function(state.num, year) {
        filename <- make_filename(year)
        data <- fars_read(filename)
        state.num <- as.integer(state.num)

        if(!(state.num %in% unique(data$STATE)))
```

```
                stop("invalid STATE number: ", state.num)
        data.sub <- dplyr::filter(data, STATE == state.num)
        if(nrow(data.sub) == 0L) {
                message("no accidents to plot")
                return(invisible(NULL))
        }
        is.na(data.sub$LONGITUD) <- data.sub$LONGITUD > 900
        is.na(data.sub$LATITUDE) <- data.sub$LATITUDE > 90
        with(data.sub, {
                maps::map("state", ylim = range(LATITUDE, na.rm = TRUE),
                          xlim = range(LONGITUD, na.rm = TRUE))
                graphics::points(LONGITUD, LATITUDE, pch = 46)
        })
}
```