```
#' Load CSV File
#'
#' @description  This is a simple function that loads a csv file (using the ¥code{filename}
#' argument) and converts it into a dataframe. An error is thrown if the file specified by
#' ¥code{filename} does not exist.
#'
#' @param filename A character string that specifies a the location and name of the file
#' to be loaded by this function
#'
#' @return This function returns a dataframe or tibble.
#'
#' @importFrom readr read_csv
#'
#' @importFrom dplyr tbl_df
#'
#' @examples
#' ¥dontrun{
#' accident_2013 <- fars_read("~/data/accident_2013.csv.bz2")
#' accident_2014 <- fars_read("~/data/accident_2014.csv.bz2")
#' accident_2015 <- fars_read("~/data/accident_2015.csv.bz2")
#' }
#'
#' @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)
}

#' Creates Filename
#'
#' @description  This is a simple function that prints "accident_<year>.csv.bz2",
#' defined by the ¥code{year} argument.
#'
#' @param year An integer number to denote the year.
#'
#' @return Returns a character string in the form of "accident_<year>.csv.bz2"
#'
#' @examples
#' ¥dontrun{
#' make_filename(2013)
#' make_filename(2014)
#' make_filename(2015)
#' }
#'
#' @export
make_filename <- function(year) {
        year <- as.integer(year)
        sprintf("accident_%d.csv.bz2", year)
}


#' Retrieves Month & Year from the Accident Year Files
#'
#' @description  This is a simple function that takes a numerical list of integers that
#' denotes ¥code{years}, accesses the accident file for each ¥code{year} of the list of
#' ¥code{years} and return a list of dataframes of ¥code{MONTH} and ¥code{year}, one
#' for each accident file, each associated with a year. The accident files for each year
#' within list must be located within the current working directory and must be named in
#' the format 'accident_<year>.csv.bz2'
#'
#' @param years A list of integer numbers, each of which denote a year.
#'
#' @return Returns a list of dataframes/tibbles of months (under the column 'MONTH') and
#' years (under the column 'year'). Each of these tibbles is associated with each accident
#' file that is associated with each of the ¥code{year} elements of the list of ¥code{years}.
#' If a ¥code{year} within the list does not have an associated file in the current working
#' directory named in the format 'accident_<year>.csv.bz2', then a warning is thrown and returns
#' NULL.
#'
#' @examples
#' ¥dontrun{}
#' fars_read_years(2013)
#' fars_read_years(list(2013, 2014))
#' fars_read_years(2013:2015)
#'
```

```
#' #A not-found warning is thrown and null is returned for the following
#' fars_read_years(list(2013, 2014, 2015, 2016))
#' fars_read_years(2017)
#' }#'
#' @export
fars_read_years <- function(years) {
        lapply(years, function(year) {
                file <- make_filename(year)
                tryCatch({
                        dat <- fars_read(file)
                        dplyr::mutate(dat, year = year) %>%
                                dplyr::select(MONTH, year)
                }, error = function(e) {
                        warning("invalid year: ", year)
                        return(NULL)
                })
        })
}

#' Count the number of accidents within each month for each year
#'
#' @description  This is a simple function that takes a list of numerical integers,
#' each element of which denotes a year, and produces a data frame with the number of accidents
#' for each month within each year.
#'
#' @param years A list/vector of integer numbers, each of which denote a year.
#'
#' @return Returns a pivot data frame containing two columns (one for accident count, and
#' one for each year in the ¥code{years} list/vector), where each month is a row. As per use
#' of the fars_read_years function, a warning will be returned if an element of ¥code{years}
#' does not have an associated file.
#'
#' @examples
#' ¥dontrun{
#' fars_summarize_years(2013)
#' fars_summarize_years(2013:2014)
#' fars_summarize_years(list(2013, 2014, 2015))
#' }
#'
#' @importFrom dplyr bind_rows %>% group_by summarize
#' @importFrom tidyr spread
#' @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)
}


#' Maps accidents onto states
#'
#' @description  Takes input of a ¥code{state.num} and ¥code{year} as arguments and plots the
#' accidents onto a map of the states. If the state number is invalid, an error is thrown.
#' If there are no accidents in that state, a message is returned that there are no
#' accidents to plot.
#'
#' @param state.num A numerical integer denoting the US state as is shown in the data set
#' @param year A numerical integer denoting the year
#'
#' @return Returns a plot of states with the number of accidents on each states
#' the accidents based on the ¥code{year}. Returns an error if ¥code{state.num} or if
#' ¥code{year} do not exist in the data set.
#'
#' @examples
#' ¥dontrun{
#' fars_map_state{20, 2013}
#' fars_map_state{10, 2014}
#' fars_map_state{30, 2016}
#'
#' fars_map_state{72, 2013} #Error because ¥code{state.num} doesn't exist
#' fars_map_state{20, 2020} #Error because ¥code{year} doesn't exist
#' }
#'
#' @import dplyr filter
#' @import maps map
#' @import graphics points
#'
#' @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)
        })
}
```