R/basic_checks.R - 98.99%

```
     ###############################################################################
     #' Function to throw error on invalid directory or file or if the file is
     #' not readable
     #' @param filename  name of a file or directory
     #' @return 0 if success, non zero negative values if failure
     #' @examples
     #' test_file_exist_read(system.file("extdata", "blank.txt",
     #' package = "valueEQ5D"))
     #' @export
     test_file_exist_read <- function(filename) {
       ## Checking if the file exists
 2x    if (file.exists(filename)) {
         ## Checking if the file is accessible to read
 1x      if (file.access(filename, 0) != 0) {
  !        stop(" Error reading file ")
         }
 1x      return(0)
       } else {
 1x      stop(" Invalid directory or file ")
       }
     }
     ################################################################################
     #' Function to check the given column exists
     #' @param column_name a column name
     #' @param data data frame
     #' @return 0 if success -1 if failure
     #' @examples
     #' check_column_exist("age", data.frame(
     #'   age = rep(20, 4), sex = rep("male", 4),
     #'   stringsAsFactors = FALSE
     #' ))
     #' @export
     check_column_exist <- function(column_name, data) {
43x    one <- toupper(colnames(data))
43x    two <- toupper(column_name)
43x    if (any(one == two)) {
11x      return(0)
       } else {
32x      return(-1)
       }
     }
     ################################################################################
     #' Function to return the column number for column name
     #' @param data a data frame
     #' @param column_name column names of the data frame
     #' @return column number, if success -1, if failure
     #' @examples
     #' get_column_no_colnames(data.frame(age = rep(20, 4),
     #' sex = rep("male", 4)), "sex")
     #' @export
     get_column_no_colnames <- function(data, column_name) {
12x    data_column_names <- toupper(colnames(data))
12x    if (any(data_column_names == toupper(column_name))) {
11x      column_no <- which(data_column_names == toupper(column_name))
11x      return(column_no)
       } else {
 1x      stop("Column name does not exist")
       }
     }
     #################################################################################
     #' Function to return frequency table
```

```
    #' @param v a vector
    #' @return frequency table
    #' @examples
    #' get_frequency_table(c(1, 1, 1, 12, 2))
    #' @export
    get_frequency_table <- function(v) {
2x    if (!is.null(v)) {
1x      res <- cbind(Freq = table(v), Cumul = cumsum(table(v)), relative =
1x                       prop.table(table(v)))
1x      scores <- rownames(res)
1x      res <- cbind(scores, res)
1x      return(res)
      } else {
1x      stop("Null vector")
      }
    }
    ################################################################################
    #' Function to return mode
    #' @param v a vector
    #' @return mode if success -1 for failure
    #' @examples
    #' get_mode_for_vec(c(1, 1, 2, 3))
    #' @export
    get_mode_for_vec <- function(v) {
7x    if (is.numeric(v)) {
5x      uniqv <- unique(v)
5x      uniqv[which.max(tabulate(match(v, uniqv)))]
      } else {
2x      stop("Non numeric data")
      }
    }
    ###############################################################################
    #' Function to check format of a numeric column when the values are not bounded
    #' @param vec a column vector
    #' @param nrcode non response code corresponding to the column
    #' @return 0, if success -1, if failure
    #' @examples
    #' test_data_num_norange(c(1, 2, 3, 4, -99), -99)
    #' @export
    test_data_num_norange <- function(vec, nrcode = NA) {
9x    entry <- vec
9x    if (is.na(nrcode)) {
5x      no_nrcode_entries <- entry[!is.na(entry)]
      } else {
4x      no_nrcode_entries <- entry[entry != nrcode & !is.na(entry)]
      }
9x    if (is.numeric(no_nrcode_entries)) {
5x      return(0)
      } else {
4x      stop("Some values-other than NR code is not numeric")
      }
    }
    ################################################################################
    #' Function to return descriptive statistics, sum, no of observations,
    #' mean, mode. median, range, standard deviation and standard error
    #' @param colum column
    #' @param column_name the column name
    #' @param nrcode non response code corresponding to the column
    #' @return the descriptive statistics for success , -1 for failure
    #' @examples
    #' descriptive_stat_data_column(c(1, 2, 3, 4, NA), "scores", NA)
    #' @import stats
    #' @export
    descriptive_stat_data_column <- function(colum, column_name, nrcode = NA) {
6x    vec <- colum
```

```
 6x    if (test_data_num_norange(vec, nrcode) == 0) {
 3x     this_column <- colum
 3x      if (is.na(nrcode)) {
 2x        this_column <- this_column[!is.na(colum)]
         } else {
 1x        this_column <- this_column[colum != nrcode & !is.na(colum)]
         }
 3x      this_sum <- sum(this_column)
 3x      this_av <- mean(this_column)
 3x      this_med <- median(this_column)
 3x      this_mode <- get_mode_for_vec(this_column)
 3x      this_range_low <- min(this_column)
 3x      this_range_high <- max(this_column)
 3x      this_sd <- sd(this_column)
 3x      this_se <- this_sd / sqrt(length(this_column))
 3x      results <- matrix(c(this_sum, this_av, this_sd, this_med, this_mode,
 3x                          this_se, this_range_low, this_range_high,
 3x                          length(this_column)), byrow = TRUE, nrow = 1)
 3x      colnames(results) <- c("Sum", "Mean", "SD", "Median", "Mode",
 3x                             "SE", "Minimum", "Maximum", "Count")
 3x      rownames(results) <- column_name
 3x      return(results)
       }
     }
     ################################################################################
     #' Function to convert a number to individual digits
     #' @param this_number a number
     #' @return digits
     #' @examples
     #' convert_number_to_digits(234)
     #' @export
     convert_number_to_digits <- function(this_number) {
 2x    string_number <- toString(this_number)
 2x    result <- suppressWarnings(as.numeric(strsplit(string_number, "")[[1]]))
 2x    if (any(is.na(result))) {
 1x      stop("The responses are not valid")
       } else {
 1x      return(result)
       }
     }
     ################################################################################
     #' Function to return the column number for a given column name
     #' (from list of possible column names that may
     #' have used) in a data frame
     #' @param column_names column names in a data frame
     #' @param data a data frame
     #' @return the column number
     #' @examples
     #' get_colno_existing_colnames(c("age"), data.frame(age = rep(20, 4),
     #' gender = rep("male", 4)))
     #' @export
     get_colno_existing_colnames <- function(column_names, data) {
12x    ans_columns <- unlist(lapply(column_names, check_column_exist, data))
12x    if (sum(ans_columns == 0) > 0) {
10x      this_col <- which(ans_columns == 0)
10x      colnum <- get_column_no_colnames(data, column_names[this_col])
10x      return(colnum)
       } else {
 2x      stop("No column exists with specified column names")
       }
     }
     ################################################################################
     #' Function to check the gender column and age column subset based on
     #' the values in it
     #' have used) in a data frame
```

```
    #' @param data a data frame
    #' @param gender groupby gender either male or female expected
    #' @param agelimit list of ages e.g. c(10,20)
    #' @return the column number
    #' @examples
    #' subset_gender_age_to_group(data.frame(age = rep(20, 4), gender =
    #' rep("male", 4)), "male", c(10, 70))
    #' @export
    subset_gender_age_to_group <- function(data, gender, agelimit) {
7x    if (is.null(gender) || toupper(gender) == "NA" || is.na(gender)) {
1x      working_data <- data   # if no groupby option given
      } else {# groupby option is given
        # groupby is male or female
6x      if (toupper(gender) == "MALE" || toupper(gender) == "FEMALE") {
4x       gendercolumn <- c("sex", "gender", "male", "female", "f", "m")
4x        colnum <- get_colno_existing_colnames(gendercolumn, data)
4x        data_gender <- unlist(data[colnum])
4x        if (toupper(gender) == "MALE") {# groupby is male
2x          malech <- c("M", "m", "male", "MALE", "Male")
2x          charinccol <- malech[malech %in% data_gender]
2x          working_data <- data[is.element(data_gender, charinccol), ]
          } else {# groupby is female
2x          femalech <- c("F", "f", "female", "FEMALE", "Female")
2x          charinccol <- femalech[femalech %in% data_gender]
2x          working_data <- data[is.element(data_gender, charinccol), ]
          }
        } else {
2x        stop("Group by should be euther male or female")
        }
      }
5x    if (is.null(agelimit) || sum(toupper(agelimit) == "NA") != 0 ||
5x        sum(is.na(agelimit)) != 0) { # no agelimit option given
2x      working_data <- working_data
      } else {# agelimit option given
3x      lowerlimit <- agelimit[1]
3x      upperlimit <- agelimit[2]
3x      age_columns <- c("age")
3x      colnum <- get_colno_existing_colnames(age_columns, working_data)
3x      working_data <- working_data[working_data[colnum] >= lowerlimit &
3x                                       working_data[colnum] <= upperlimit, ]
      }
5x    return(working_data)
    }
    ###############################################################################
    #' Function to add an underscore for texts with spaces in between
    #' @param this_string a string
    #' @return  string where the spaces replaced by "_"
    #' @examples
    #' replace_space_underscore("Sri Lanka")
    #' @export
    replace_space_underscore <- function(this_string) {
3x    sep_string <- unlist(strsplit(this_string, " "))
3x    if (length(sep_string) < 1) {
1x      stop("Error in separating the string")
      } else {
2x      new_string <- sep_string[1]
2x      if (length(sep_string) > 1) {
1x        for (i in 2:length(sep_string)) {
1x          new_string <- cbind(new_string, sep_string[i])
          }
1x        new_string <- paste(new_string, collapse = "_")
        } else {
1x        new_string <- sep_string
        }
2x      return(new_string)
```

```
  }
}
```