functions.R

```
##' Inserts escape characters to protect preserved symbols in LaTeX
##' markup
##'
##' Found fixes for all of the LaTeX symbols except backslash
##' @param x a string
##' @return corrected character vector for LaTeX
##' @author Paul Johnson
##' @examples
##' x <- c("_asdf&_&$", "asd adf asd_", "^ % & $asdf_")
##' escape(x)
escape <- function (x){
    x <- gsub("\\\\", "SANITIZE.BACKSLASH", x)
    for(i in c("&", "$", "_", "^", "%", "#", "{", "{")) {
        x <- gsub(paste0("\\", i), paste0("\\\\",i), x)
    }
    x <- gsub("\\~", "$\\sim$", x)
    x <- gsub(">", "$>$", x, fixed = TRUE)
    x <- gsub("<", "$<$", x, fixed = TRUE)
    x <- gsub("|", "$|$", x, fixed = TRUE)
    x <- gsub("^", "\\verb|^|", x, fixed = TRUE)
    x <- gsub("SANITIZE.BACKSLASH", "$\\backslash$",
              x, fixed = TRUE)
    x
}

##' Retrieve column names required from QC
##'
##' @param datadir Directory name, ends in "/"
##' @param fn file name
##' @return vector of column names
##' @author Paul Johnson
getQC <- function(datadir, fn = "QC_Required_Columns.xlsx"){
    require(openxlsx)
    qc <- read.xlsx(paste0(datadir, "/", fn), colNames = TRUE)
    colnames(qc)
}

## dts: delete trailing slash
## Will delete repeat slashes first as well
dts <- function(name) gsub("/$", "", dms(name))

##' Replace multiple slashes with one slash
dms <- function(name) gsub("(/)\\1+", "/", name)


keywrds <- c("District", "County",  "Middle.School", "High.School", "School",
             "Elementary", "Northeast", "Southwest", "Northwest", "Southeast",
             "Academy", "Cooperative", "Regional", "Foundations", "Preparatory",
             "Community", "For.The.Blind.and.Visually.Impaired", "Independent",
             "Special.Education",
             "DISTRICT", "COUNTY", "SCHOOL", "ELEMENTARY", "MIDDLE.SCHOOL",
             "HIGH.SCHOOL", "NORTHEAST", "SOUTHWEST", "NORTHWEST", "SOUTHEAST",
             "ACADEMY", "COOPERATIVE", "REGIONAL", "FOUNDATIONS", "PREPARATORY",
             "COMMUNITY", "FOR.THE.BLIND.AND.VISUALLY.IMPAIRED", "INDEPENDENT",
             "SPECIAL.EDUCATION")


```

```
keyabr <- c("Dist", "Co", "MS", "HS", "Schl",
             "Elem", "NE", "SW", "NW", "SE",
             "Acdmy", "Co-op", "Reg", "Found", "Prep",
             "Comm", "For_BVI", "Ind", "SPED",
             "DIST", "CO", "SCHL", "ELEM", "MS",
             "HS", "NE", "SW", "NW", "SE",
             "ACDMY", "CO-OP", "REG", "FOUND", "PREP",
             "COMM", "FOR_BVI", "IND", "SPED")

names(keywrds) <- keyabr
states.long <- c(state.name[48], state.name[-48], toupper(state.name)[48], toupper
    (state.name)[-48], "Choctaw", "Miccosukee")
states.short <- c(state.abb[48], state.abb[-48], state.abb[48],  state.abb[-48], "
    Choctaw", "Miccosukee")
names(states.short) <- states.long

abbr <- function(x, long, short = NULL){
    if (is.null(short)) short = names(long)
    for (i in seq_along(long)){
        x <- gsub(long[i], short[i], x)
    }
    x
}

shorten <- function(x, k = 20){
    y <- substr(x, 1, k)
    make.unique(y)
}

shrt <- function(x, k = 20) shorten(abbr(abbr(x, states.long, states.short),
    keywrds), k = 20)

abbrState1st <- function(x, state, long = states.long, short = states.short){
    if (is.null(names(short))) stop("short name is unnamed")
    xx <-  ifelse(!is.null(short[state]), gsub(paste0("^", state), short[state], x
        ), x)
    xx
}


##' remove non-alpha numeric characters
##'
##' regular expression matching
##' @param x
##' @return
##' @author Paul Johnson
clnstrng <- function(x){
    y <- gsub("[^a-zA-Z0-9\\\\' _()-]", "", x)
    y <- gsub("\\ *$", "", y) # Trailing spaces
    y
}

## clean file name string component
cfn <- function(x){
    xn <- gsub("\\* $", "", x)
    xn <- gsub("\\", "", x, fixed = TRUE)
    xn <- gsub("\\ ", "_", xn)
    xn <- gsub("_-_", "-", xn)
    xn <- gsub("_-", "-", xn)
```

```
    xn <- gsub("-_", "-", xn)
    xn <- gsub("&", "and", xn)
    xn <- gsub("\\\'","", xn)
    xn <- gsub("\\\"","", xn)
    xn <- gsub("#", "_", xn, fixed = TRUE)
    xn <- gsub("c/o", "co", xn)
    xn <- gsub("/", "-", xn)
    xn <- gsub("\\*", "", xn)
    xn <- gsub("\\$", "", xn)
    xn <- gsub("-$", "",  xn)
    xn <- gsub("_$", "", xn)
    xn
}

## recursively remove null objects
## See: http://stackoverflow.com/questions/26539441/r-remove-null-elements-from-list-of-lists
is.NullOb <- function(x) is.null(x) | all(sapply(x, is.null))

## Recursively step down into list, removing all such objects
rmNullObs <- function(x) {
   x <- Filter(Negate(is.NullOb), x)
   lapply(x, function(x) if (is.list(x)) rmNullObs(x) else x)
}



##' Remove rows that are more than 90% NA
##'
##' Some imported Excel sheets have "noise" in the last
##' rows, which appear as <NA> or NA in R. Get rid of them.
##' @title
##' @param dframe
##' @return
##' @author Paul Johnson
deleteBogusRows <- function (dframe){
    rowna <- apply(dframe, 1, function(x){sum(is.na(x))})
    badrows <- rowna > .9 * dim(dframe)[2]
    if (any(badrows)){
        cat(paste("deleteBogusRows Diagnostic\n"))
        cat(paste("These rows from the data frame: ", deparse(substitute(dframe)),
             "\n are being purged:"))
        print(rownames(dframe)[badrows])
        cat(paste("The bad content was\n"))
        print(dframe[badrows, ])
        newdframe <- dframe[!badrows,]
        return(newdframe)
    }
    print(paste("No bogus rows were found in: ", deparse(substitute(dframe))))
    invisible(dframe)
}



##' Searches a data frame to find column name components that match at least one element in a specified set of characters
##'
```

```
169 ##'
170 ##' @param colnames A vector of words or symbols to use when searching
171 ##'     for matches
172 ##' @param data A data frame with named columns to be searched
173 ##' @param lower Logical. Determines whether or not the tolower()
174 ##'     function is used to ensure that all column names and search
175 ##'     elements match in terms of case
176 ##' @param names Logical. Determines whether or not the names of the
177 ##'     columns are returned by the function.  The default is to
178 ##'     simply return the column numbers.
179 ##' @return Returns either the names of the columns, or the column
180 ##'     numbers that match at least one element in colnames.
181 ##' @author Ben Kite
182 cs <- function(strngs, data, lower = TRUE, names = TRUE){
183     names.orig <- names(data)
184     if (lower == TRUE){
185         varlist <- tolower(names.orig)
186         strngs <- tolower(strngs)
187     }else{
188         varlist <- names.orig
189         strngs <- strngs
190     }
191
192     varlist.orig <- varlist
193     ##  whittle down varlist candidates iteratively (lexicographic search)
194     for (i in strngs){
195         varlist <- varlist[grep(i, varlist)]
196     }
197
198     if (length(varlist) > 1) stop(paste("There is more than one candidate in cs",
        varlist))
199     winner <- match(varlist, varlist.orig)
200
201     if (names) return(names.orig[winner]) else winner
202 }
203
204
205 ##' Insert 0's in the front of existing digits or characters so that
206 ##' all elements of a vector have the same number of characters.
207 ##'
208 ##' The main purpose was to correct ID numbers in studies that are
209 ##' entered as integers with leading 0's as in 000001 or 034554.  R's
210 ##' default coercion of integers will throw away the preceding 0's,
211 ##' and reduce that to 1 or 34554. The user might want to re-insert
212 ##' the 0's, thereby creating a character vector with values "000001"
213 ##' and "045665".
214 ##'
215 ##' If x is an integer or a character vector, the result is the
216 ##' more-or-less expected outcome (see examples). If x is numeric,
217 ##' but not an integer, then x will be rounded to the lowest integer.
218 ##'
219 ##' The previous versions of this function failed when there were
220 ##' missing values (NA) in the vector x.  This version returns NA for
221 ##' those values.
222 ##'
223 ##' One of the surprises in this development was that sprintf() in R
224 ##' does not have a known consequence when applied to a character
225 ##' variable. It is platform-dependent (unredictable). On Ubuntu Linux
226 ##' 16.04, for example \code{sprintf("\%05s", 2)} gives back
```

```
227 ##' \code{" 2"}, rather than (what I expected) \code{"00002"}. The
228 ##' problem is mentioned in the documentation for \code{sprintf}. The
229 ##' examples show this does work now, but please test your results.
230 ##' @param x vector to be converted to a character variable by
231 ##'     inserting 0's at the front. Should be integer or character,
232 ##'     floating point numbers will be rounded down. All other
233 ##'     variable types will return errors.
234 ##' @param n Optional parameter.  The desired final length of
235 ##'     character vector.  This parameter is a guideline to determine
236 ##'     how many 0's must be inserted.  This will be ignored if
237 ##'     \code{n} is smaller than the number of characters in the
238 ##'     longest element of \code{x}.
239 ##' @return A character vector
240 ##' @author Paul Johnson <pauljohn@@ku.edu>
241 ##' @examples
242 ##' x1 <- c(0001, 0022, 3432, NA)
243 ##' padW0(x1)
244 ##' padW0(x1, n = 10)
245 ##' x2 <- c("1", "22", "3432", NA)
246 ##' padW0(x2)
247 ##' ## There's been a problem with strings, but this works now.
248 ##' ## It even preserves non-leading spaces. Hope that's what you want.
249 ##' x3 <- c("1", "22 4", "34323  42", NA)
250 ##' padW0(x3)
251 ##' x4 <- c(1.1, 334.52, NA)
252 ##' padW0(x4)
253 padW0 <- function (x, n = 0) {
254     if (!is.numeric(x) && !is.character(x))
255         stop("padW0 only accepts integer or character values for x")
256
257     xismiss <- is.na(x)
258     ## springf trouble with characters, not platform independent
259     if(is.numeric(x)) {
260         if (is.double(x)) x <- as.integer(x)
261         maxlength <- max(nchar(x), n, na.rm = TRUE)
262         vtype <- "d"
263         res <- sprintf(paste0("%0", maxlength, vtype), x)
264     } else {
265         maxlength <- max(nchar(x), n, na.rm = TRUE)
266         xlength <- vapply(x, nchar, integer(1))
267         padcount <- maxlength - xlength
268         pads <- vapply(padcount, function(i){paste0(rep("0", max(0, i, na.rm=TRUE)
              ), collapse = "")}, character(1))
269         res <- paste0(pads, x)
270     }
271
272     res[xismiss] <- NA
273     res
274 }
275
276
277
278
279
280 ##' Cuts a string at a specified linewidth, trying to align cut with a
281 ##' separator
282 ##'
283 ##' Some strings are simply too long. Let's shorten them and attempt
284 ##' to cut at a specified separator. Use vectorized function,
```

```
##' shortenv, to apply to non-scalar
##' @param textstring target string to shorten
##' @param linewidth approximate length to shorten taking into account
##'     tolerance
##' @param tol number of characters forward/back to check; if single
##'     value then only backwards checking
##' @param capwidth integer specifying the width of capital letters
##' @param separator accepts 1- or 2-element vector to separate
##'     string; default = c(" ", "_")
##' @return shortened character string
##' @author Brent Kaplan, Ben Kite, Paul Johnson
##' @examples
##' test <- "123_567_9"
##' (truncsmart(test, 5, tol = c(1,2)))
##'
truncsmart <- function(textstring, linewidth, tol = c(5, 5),
                       capwidth = 1.2, separator = c(" ", "_")) {
    if(length(tol) > 2) stop("Please specify 1 or 2 values for tol.")
    lets <- unlist(strsplit(textstring, split = ""))
    caps <- sapply(lets, function(x) x %in% LETTERS)
    widthval <- ifelse(caps == TRUE, capwidth, 1)
    linelength <- cumsum(widthval)
    if(linelength[length(linelength)] <= linewidth) return(textstring)
    withinlength <- linelength  <= linewidth
    ifelse(length(tol) == 2, withintol <- linelength <= (linewidth + tol[2]),
           withintol <- linelength <= (linewidth + tol))
    letswl <- lets[withinlength]
    letso <- letswl
    letswt <- lets[withintol]
    if(length(tol) == 2) {
        for (j in 0:tol[1]){
            if(identical(letswl[length(letswl)], separator[1]) ||
               identical(letswl[length(letswl)], separator[2])) {
                letso1 <- letswl[-length(letswl)]
                break()
            }
            letswl <- letswl[-length(letswl)]
        }
        for (j in 0:tol[2]) {
            if(identical(letswt[length(letswt)], separator[1]) ||
               identical(letswt[length(letswt)], separator[2])) {
                letso2 <- letswt[-length(letswt)]
                break()
            }
            letswt <- letswt[-length(letswt)]
        }
        ifelse(!exists("letso1") && !exists("letso2"), letso,
        ifelse(exists("letso1") && !exists("letso2"), letso <- letso1,
        ifelse(!exists("letso1") && exists("letso2"), letso <- letso2,
        ifelse(abs((length(letso1) - length(lets[withinlength]))) <= (length(
            letso2) - length(lets[withintol])), letso <- letso1, letso <- letso2)))
            )
    } else {
        for (j in 0:tol){
            if(identical(letswl[length(letswl)], separator[1]) ||
               identical(letswl[length(letswl)], separator[2])) {
                letso <- letswl[-length(letswl)]
                 break()
            }
```

```
            letswl <- letswl[-length(letswl)]
        }
    }
    out <- paste0(letso, collapse  = "")
    out
}
truncsmart <- Vectorize(truncsmart, USE.NAMES=FALSE)



##' Initial letter will be capitalized
##'
##' copied from Frank Harrell's Hmisc package
##' @param string argument to be capitalized
##' @return A capitalized string
##' @author Frank Harrell
capitalize <- function (string)
{
    capped <- grep("^[^A-Z]*$", string, perl = TRUE)
    substr(string[capped], 1, 1) <- toupper(substr(string[capped],
        1, 1))
    return(string)
}



##' Cuts a string at a specified linewidth, trying to align cut with a separator
##'
##' Some strings are simply too long. Let's shorten them and attempt
##' to cut at a specified separator
##' @param textstring
##' @param linewidth
##' @param separator defaults to space but can be underscore "_"
##' @return shortened character string
##' @author Ben Kite, Paul Johnson, Brent Kaplan
##' @examples
##' test <- "An example string that is way too long to be useful and therefore
   needs to be cut down to our specified linewidth."
##' splitter(test, 50)
##'
splitter <- function(textstring, linewidth, separator = " "){
    lets <- unlist(strsplit(textstring, split = ""))
    caps <- sapply(lets, function(x) x %in% LETTERS)
    widthval <- ifelse(caps == TRUE, 1.2, 1)
    linelength <- cumsum(widthval)
    if(linelength[length(linelength)] < linewidth) return(textstring)
    withinlength <- linelength  < linewidth
    lets2 <- lets[withinlength]
    lets3 <- lets2
    for (j in 1:5){
        if(lets3[length(lets3)] == separator) {
            lets3 <- lets3[-length(lets3)]
            lets2 <- lets3
            break()
        }
        lets3 <- lets3[-length(lets3)]
    }
    out <- paste0(lets2, collapse  = "")
    out
```

```
}


##' Replace column names with new names from a named vector
##'
##' Imagine some beautiful writing here
##' @param dat a data frame
##' @param newname A named vector of the form c(oldname1 = "newname1", oldname2 = "newname")
##' @param lowercase should new names be converted to lower case?
##' @return a data frame
##' @export
##' @author Paul Johnson <pauljohn@@ku.edu>
colnamesReplace <- function(dat, newname, lowercase = FALSE){
    if (is.null(names(newname))){
        stop("The newname vector should be a named vector, using the oldnames as the names")
    }
    oldname <- colnames(dat)
    if(any(newname[oldname] != oldname)){
        cat(paste("colnamesReplace Diagnostic: These names are to be replaced:\n"))
        res <- ifelse (oldname %in% names(newname), newname[oldname], oldname)
        print(c("oldname", "newname"))
        print(cbind(oldname, res))
        colnames(dat) <- if (lowercase) tolower(res) else res
        return(dat)
    } else{
        cat(paste("replaceNames Diagnostic:  No column names were altered"))
        return(dat)
    }
}
```