internalFunctions.r

ella

Wed May 09 11:58:39 2018

```
 # ---------------------------------------------------------------------------
# fixData function ----------------------------------------------------------
```

Fixes input CSV data (e.g. takes care of factors, and other data frame conversions) @keywords internal @param dta any data frame @param compbl compress multiple blank spaces to single blank space for all character or factor variables in the dataset

```
fixData <- function(dta,compbl=FALSE) {
  dta<-as.data.frame(dta) # have to convert to dataframe from local dataframe from readxl
  # run through the data
  colnames(dta) <- tolower(trimws(colnames(dta)))

  # remove rows that have all NAs (EM)
  countnas=as.numeric(apply(data.frame(dta),1,function(x) length(which(is.na(x)))))
  if (length(which(countnas==ncol(dta)))>0) {
    dta=dta[-c(which(countnas==ncol(dta))),]
  }

  cls <- sapply(dta, class)
  # do conversions for data types: integer to numeric and dates are identified by date in name

  if (length(which(cls == "integer")) > 0) {
    for (ind in which(cls == "integer"))
      dta[,ind] <- as.numeric(dta[,ind])
  }

  if (length(which(cls == "factor")) > 0) {
    for (ind in which(cls == "factor"))
      dta[,ind] <- trimws(as.character(dta[,ind])) # trim and convert to character
  }
  if (length(which(cls == "character")) > 0) {
    for (indc in names(dta)) {
      if(class(dta[, indc]) %in% c("factor", "character")){
        if (compbl==TRUE)
          dta[, indc] <- gsub("\\s+", " ",trimws(dta[, indc])) # compress duplicate blanks
        else
          dta[, indc] <- trimws(dta[, indc])
      }
    }

  }

  return(dta)
} # end fixData function
```

```
# checkIntegrity function ---------------------------------------------------
```

Checks integrity of sheets in the user input CSV file @keywords internal @param dta.metab dta.metab

@param dta.smetab dta.smetab @param dta.sdata dta.sdata @param dta.vmap dta.vmap @param dta.models dta.models

```
checkIntegrity <- function (dta.metab,dta.smetab, dta.sdata,dta.vmap,dta.models) {

    print("Running Integrity Check...")
    # get the cohort equivalent of metabolite_id and subject id
    metabid = tolower(dta.vmap$cohortvariable[tolower(dta.vmap$varreference) == "metabolite_id"])
    subjid = tolower(dta.vmap$cohortvariable[tolower(dta.vmap$varreference) == 'id'])
    # add _ to all metabolites before splitting at blank
    allmodelparams=c(dta.models$outcomes,dta.models$exposure, dta.models$adjustment,dta.models$stratific
    allmodelparams=gsub("All metabolites","All_metabolites",gsub("\\s+", " ", allmodelparams[!is.na(allm
    print(paste(dta.models$ccovs,dta.models$scovs))

    # take out multiple blanks and add _ to all metabolites to avoid splitting
    allmodelparams=tolower(unique(unlist(stringr::str_split(allmodelparams," "))))
    outmessage = c()
    if (length(metabid) == 0) {
      stop("metabid is not found as a parameter in VarMap sheet!  Specify which column should be used fo
    }
    else if (!is.na(dta.models$stratification) &&
        length(intersect(dta.models$adjustment,dta.models$stratification))>=1) {
    stop("Adjustment and stratification parameters are the same!  This is not allowed.")
    }
    else if (!is.na(dta.models$stratification) &&
        length(intersect(dta.models$exposure,dta.models$stratification))>=1) {
        stop("Exposure and stratification parameters are the same!  This is not allowed.")
    }
    else if (length(intersect(allmodelparams,
         tolower(c("All_metabolites",  dta.vmap$varreference)))) !=length(allmodelparams))
# tolower(c("All metabolites", colnames(dta.smetab), colnames(dta.sdata)))))!=length(allmodelparams))
{
         stop("Parameters in model data ('Models' sheet in input file) do not exist!  Check the naming!
    }
    else if (length(subjid) == 0) {
        stop("id (for subject id) is not found as a parameter in VarMap sheet!  Specify which column sho
    }
    else if (length(intersect(subjid,colnames(dta.sdata))) != 1) {
        stop("The user input id in the 'COHORTVARIABLE' column of the Varmap Sheet is not found in the
    }
    else if (length(intersect(subjid,colnames(dta.smetab))) != 1) {
        stop("The user input id in the 'COHORTVARIABLE' column of the Varmap Sheet is not found in the
    }
    else if (length(intersect(metabid,colnames(dta.metab))) != 1) {
        stop("The user input metabolite_id in the 'COHORTVARIABLE' column of the Varmap Sheet is not fou
    }
    else {
      #print("Passed the checks")
      dta.metab[[metabid]] = tolower(dta.metab[[metabid]])
      dta.sdata[[subjid]] = tolower(dta.sdata[[subjid]])
      dta.smetab[[subjid]] = tolower(dta.smetab[[subjid]])
      if (length(grep(metabid,colnames(dta.metab))) == 0) {
          stop("Error: Metabolite ID from 'VarMap Sheet' (",metabid,") does not match column name from
      }
      else if (length(grep(subjid,colnames(dta.sdata))) == 0) {
          stop("Error: Sample ID from 'VarMap Sheet' (",subjid,") does not match a column name in 'Subje
      }
      else if (length(unique(dta.sdata[,subjid])) != length(unique(dta.smetab[,subjid]))) {
        outmessage = c(
          outmessage,"Warning: Number of subjects in SubjectData sheet does not match number of subjects
        )
      }
      else if (length(unique(colnames(dta.smetab))) != ncol(dta.smetab)) {
        outmessage = c(
          outmessage,"Warning: Metabolite abundances sheet (SubjectMetabolites) contains duplicate colum
        )
      }
      else if (length(unique(unlist(dta.sdata[,subjid]))) != nrow(dta.sdata)) {
        outmessage = c(
          outmessage,"Warning: Sample Information sheet (SubjectData) contains duplicate ids"
        )
      }
      else if (length(unique(unlist(dta.metab[,metabid]))) != nrow(dta.metab)) {
        outmessage = c(
          outmessage,"Warning: Metabolite Information sheet (Metabolites) contains duplicate metabolite
        )
      }
      else {
        nummetab = length(unique(colnames(dta.smetab)[-c(which(colnames(dta.smetab) ==
                                                              subjid))]))
        numsamples = length(unique(dta.smetab[[subjid]]))
        if (length(intersect(as.character(unlist(dta.metab[,metabid])),colnames(dta.smetab)[-c(which(col
                                                                                                      sub
            length(intersect(as.character(unlist(dta.sdata[,subjid])),dta.smetab[[subjid]])) ==
            numsamples) {
          outmessage = c(
            outmessage,"Passed all integrity checks, analyses can proceed. If you are part of COMETS, pl
          )
        }
        else {
          if (length(intersect(tolower(make.names(dta.metab[[metabid]])),tolower(colnames(dta.smetab)))
              nummetab) {
              stop("Error: Metabolites in SubjectMetabolites DO NOT ALL match metabolite ids in Metaboli
          }
          if (length(intersect(dta.sdata[[subjid]],dta.smetab[[subjid]])) !=
              numsamples) {
              stop("Error: Sample ids in SubjectMetabolites DO NOT ALL match subject ids in SubjectData
          }
        }
      }
    }

   ########################################
   # Check that models are reasonable
   ########################################
   # Check that adjustment variables that at least two unique values
##   for (i in dta.models$adjustment) {
```

```
##        temp <- length(unique(dta.sdata[[i]]))
##  if(temp <= 1 && !is.na(i)) {
##      outmessage<-c(outmessage,paste("Error: one of your models specifies",i,"as an adjustment but tha
##          one possible value"))
##      }
##   }
##
##   # Check that stratification variables that at least two unique values
##   for (i in dta.models$stratification) {
##        temp <- length(unique(dta.sdata[[i]]))
##        if(temp <= 1 && !is.na(i)) {
##                outmessage<-c(outmessage,paste("Error: one of your models specifies",i,"as an stratifi
##                        one possible value"))
##        }
##   }

    if (is.null(outmessage)) {
      outmessage = "Input data has passed QC (metabolite and sample names match in all input files)"
    }

    return(
      list(
        dta.smetab = dta.smetab,dta.metab = dta.metab, dta.sdata = dta.sdata,outmessage =
          outmessage
      )
    )
  } # end checkIntegriy
```

```
# Harmonize ---------------------------------------------------
```

Fixes input CSV data (e.g. takes care of factors, and other data frame conversions) @keywords internal @param dtalist results of reading a CSV data sheet (with read_excel)

```
Harmonize<-function(dtalist){
  mastermetid=metabolite_name=metlower=uid_01=cohorthmdb=foundhmdb=masterhmdb=NULL
  # Load processed UIDs file:
  dir <- system.file("extdata", package="COMETS", mustWork=TRUE)
  masterfile <- file.path(dir, "compileduids.RData")
  load(masterfile)

  # rename metid to be the same as metabid
  colnames(mastermetid)[which(colnames(mastermetid)=="metid")]=dtalist$metabId


  # join by metabolite_id only keep those with a match
  harmlistg<-dplyr::inner_join(dtalist$metab,mastermetid,by=c(dtalist$metabId),suffix=c(".cohort",".come

  # Loop through and try to join all the other columns (at each loop, combine matches and remove
  # non-unique entries
  for (i in setdiff(colnames(dtalist$metab),dtalist$metabId)) {
    harmlistg<-rbind(harmlistg,
        dplyr::left_join(
            dplyr::anti_join(dtalist$metab,harmlistg,
                    by=c(dtalist$metabId)) %>%
                     dplyr::mutate(metlower=gsub("\\*$","",i)),
                mastermetid,by=c("metlower"=dtalist$metabId),suffix=c(".cohort",".comets")) %>%
        dplyr::select(-metlower)) #%>%
#       dplyr::mutate(multrows=grepl("#",uid_01),harmflag=!is.na(uid_01))
  }

  # join by metabolite_name only keep those with a match
#  harmlistc<-dplyr::left_join(dplyr::anti_join(dtalist$metab,mastermetid,
#        by=c(dtalist$metabId)) %>%
#          dplyr::mutate(metlower=gsub("\\*$","",tolower(metabolite_name))), # take out * in metabolite
 #       mastermetid,by=c("metlower"=dtalist$metabId)) %>% dplyr::select(-metlower)

  # combine the 2 data frames
#  dtalist$metab<-rbind(harmlistg,harmlistc) %>%
#    dplyr::mutate(multrows=grepl("#",uid_01),harmflag=!is.na(uid_01))

# Reorder:
  myord <- as.numeric(lapply(dtalist$metab[,dtalist$metabId],function(x)
    which(harmlistg[,dtalist$metabId]==x)))
  finharmlistg <- harmlistg[myord,]

# routine for hmdb look-up for those without a match
  if (length(names(finharmlistg)[grepl("^hmdb",names(finharmlistg))])>=2){

    # first hmdb is from cohort metabolite metadata
    cohorthmdb <- names(finharmlistg)[grepl("^hmdb",names(finharmlistg))][1]

    # need to rename to hmdb_id so that it can be left_join match
    names(finharmlistg)<-gsub(cohorthmdb,"hmdb_id",names(finharmlistg))

    # bring in the masterhmdb file to find further matches
    foundhmdb<-finharmlistg %>%
      filter(is.na(uid_01)) %>% # only find match for unmatched metabolites
      select(1:ncol(dtalist$metab)) %>%  # keep only original columns before match
      left_join(masterhmdb,suffix=c(".cohort",".comets"))

    # rename back so we can combine
    names(foundhmdb)<-gsub("hmdb_id",cohorthmdb,names(foundhmdb))
    names(finharmlistg)<-gsub("hmdb_id",cohorthmdb,names(finharmlistg))

    finharmlistg<-finharmlistg %>%
      filter(!is.na(uid_01)) %>% # take the ones with the previous match
      union_all(foundhmdb)       # union with the non-matches

    # fix found hmdb name
    names(finharmlistg)<-gsub(".cohort.comets",".comets",names(finharmlistg))

  }


  if(all.equal(sort(finharmlistg[,dtalist$metabId]),sort(dtalist$metab[,dtalist$metabId]))) {
    dtalist$metab <- finharmlistg
    return(dtalist)
  }
  else {
    stop("Something went wrong with the harmonization")
  }

}
```

```
# prdebug ---------------------------------------------------
```

debug by printing object with time time @keywords internal @param lab label of object @param x object

```
#
prdebug<-function(lab,x){
  print(paste(lab," = ",x," Time: ",Sys.time()))
}
```

Function that subsets input data based on "where variable"

@param readData list from readComets @param where users can specify which subjects to perform the analysis by specifying this parameter. 'where' expects a vector with a variable name, a comparison operator ("<", ">", "=","<=",">="), and a value. For example, "where = c("Gender=Female")". Multiple where statements should be comma separated (a vector). @return filtered list

```
filterCOMETSinput <- function(readData,where=NULL) {
  if (!is.null(where)) {
    samplesToKeep=c()
    myfilts <- strsplit(where,",")
    # create rules for each filter
    for (i in 1:length(myfilts)) {
        myrule <- myfilts[[i]]
                if(length(grep("<=",myrule))>0) {
                        mysplit <- strsplit(myrule,"<=")[[1]]
                        samplesToKeep <- c(samplesToKeep,
                           which(as.numeric(readData$subjdata[,gsub(" ","",mysplit[1])]) <= gsub(" ",""
                } else if(length(grep(">=",myrule))>0) {
                        mysplit <- strsplit(myrule,">=")[[1]]
                        samplesToKeep <- c(samplesToKeep,
                           which(as.numeric(readData$subjdata[,gsub(" ","",mysplit[1])]) >= gsub(" ",""
                } else if(length(grep("<",myrule))>0) {
          mysplit <- strsplit(myrule,"<")[[1]]
                  samplesToKeep <- c(samplesToKeep,
                           which(as.numeric(readData$subjdata[,gsub(" ","",mysplit[1])]) < gsub(" ","",
          } else if(length(grep(">",myfilts[i]))>0) {
              mysplit <- strsplit(myrule,">")[[1]]
                        samplesToKeep <- c(samplesToKeep,
                           which(as.numeric(readData$subjdata[,gsub(" ","",mysplit[1])]) > gsub(" ","",
        } else if (length(grep("=",myfilts[i]))>0) {
          mysplit <- strsplit(myrule,"=")[[1]]
                        samplesToKeep <- c(samplesToKeep,
                           which(as.numeric(readData$subjdata[,gsub(" ","",mysplit[1])]) == gsub(" ",""
          } else
              stop("Make sure your 'where' filters contain logicals '>', '<', or '='")
        }
    mycounts <- as.numeric(lapply(unique(samplesToKeep),function(x)
        length(which(samplesToKeep==x))))
    fincounts <- which(mycounts == length(myfilts))
        readData$subjdata <- readData$subjdata[unique(samplesToKeep)[fincounts],]
  }
  else {(warning("No filtering was performed because 'where' parameter is NULL"))}
return(readData)
}
```

Preprocess design matrix for zero variance, linear combinations, and dummies @keywords internal @param modeldata (output of function getModelData()) @param crateDummies TRUE or FALSE @return modeldata after checks are performed

```
checkModelDesign <- function (modeldata=NULL, createDummies=NULL) {
    if(is.null(modeldata) || is.null(createDummies)) {
        stop("Please make sure that modeldata and createDummies are defined")
    }
  errormessage=warningmessage=c()
 # Check that there are at least 15 samples (n>15)
  if (nrow(modeldata$gdta)<15){
    if (!is.null(modeldata$scovs)){
      #warning(paste("Data has < 15 observations for strata in",modeldata$scovs))
      mycorr=data.frame()
      attr(mycorr,"ptime")="Processing time: 0 sec"
      return(mycorr)
    } else{
      stop(paste(modeldata$modlabel," has less than 15 observations and will not be run."))
    }
  }

   # Check that adjustment variable that at least two unique values
   for (i in modeldata$acovs) {
        temp <- length(unique(modeldata$gdta[[i]]))
        if(temp <= 1 && !is.na(i)) {
               warning(paste("Warning: one of your models specifies",i,"as an adjustment value
               but that variable only has one possible value.
               Model will run without",i,"as an adjustment"))
               modeldata$acovs <- setdiff(modeldata$acovs,i)
        }
   }
   if (length(modeldata$acovs)==0) {
    modeldata$acovs=NULL
    createDummies=FALSE
   }

   if(createDummies) {
    print("Creating dummies")
    metabid=uid_01=biochemical=outmetname=outcomespec=exposuren=exposurep=metabolite_id=c()
    cohortvariable=vardefinition=varreference=outcome=outcome_uid=exposure=exposure_uid=c()
    metabolite_name=expmetname=exposurespec=c()

    # column indices of row/outcome covariates
    col.rcovar <- match(modeldata[["rcovs"]],names(modeldata[["gdta"]]))

    # column indices of column/exposure covariates
    col.ccovar <- match(modeldata[["ccovs"]],names(modeldata[["gdta"]]))

    # column indices of adj-var
```

```
col.adj <- match(modeldata[["acovs"]],names(modeldata[["gdta"]]))

# Defining global variable to remove R check warnings
corr=c()
loadNamespace("caret") #need this to avoid problem of not finding contr.ltfr

# Create dummy variables
myformula <- paste0("`",colnames(modeldata$gdta)[col.rcovar], "` ~ ",paste(colnames(modeldata$gdta)
dummies <- caret::dummyVars(myformula, data = modeldata$gdta,fullRank = TRUE)
mydummies <- stats::predict(dummies, newdata = modeldata$gdta)

# Check for zero-variance predictors (e.g. a stratified group that only has 1 value)
nonzero <- caret::nearZeroVar(mydummies,freqCut = 95/5)
if(length(nonzero)>0) {
        filtdummies <- mydummies[,-nonzero]
    warningmessage <- c(warningmessage,
        paste0("Removed ",paste(colnames(mydummies)[unique(nonzero)],collapse=","),
            " because of zero-variance",collapse=""))
} else {
        filtdummies <- mydummies
}

# Check for correlated predictors (this will remove the first "factor" that is highly
# correlated with another
if (ncol(filtdummies)>1){
    cors <- caret::findCorrelation(stats::cor(filtdummies,method="spearman"), cutoff = .97)
  if(length(cors)>0) {
        filtdummies2 <- filtdummies[,-cors]
    warningmessage <- c(warningmessage,
                    paste("Removed ",paste(colnames(mydummies)[unique(cors)],collapse=","),
            " because of correlation > 0.97 with other covariates",collapse=""))
}
    else {
      filtdummies2=filtdummies
    }

# Check for linear dependencies
ldeps <- caret::findLinearCombos(filtdummies2)
if(length(ldeps$remove)>0) {
        findummies <-filtdummies2[,-ldeps$remove]
    warningmessage <- c(warningmessage,
                    paste0("Removed ",paste(colnames(mydummies)[unique(ldeps$remove)],collapse=",")
            " because of linear dependencies",collapse=""))
} else
        findummies=filtdummies2
}
else {
  findummies=filtdummies
}



newdat <- cbind(modeldata$gdta[,modeldata$rcovs],findummies)
```

```
    colnames(newdat)[1:length(modeldata$rcovs)]=modeldata$rcovs
    modeldata$acovs <- grep(paste(modeldata$acovs,collapse="|"),colnames(findummies), value=TRUE)
    modeldata$ccovs <- grep(paste(modeldata$ccovs,collapse="|"),colnames(findummies), value=TRUE)
    modeldata$gdta <- newdat
     }

     print(warningmessage)
     return(list(warningmessage=warningmessage,errormessage=errormessage,modeldata=modeldata))
}
```