Doing exploratory analysis in R with a package Explorer v. 1.0

Jaroslav Flegr[a] & Pavel Flegr

[a] Division of Biology, Faculty of Science, Charles University in Prague, Prague, Viničná 7, 128 44, Czech Republic

Any confirmatory analysis should be followed by an exploratory analysis of data, i.e. by searching for interesting patterns, e.g. for an association between binary or categorical variables or a correlation between continuous variables. When the data set is large and contains many variables, this part of the analysis can be rather laborious. The new package Explorer makes such analysis much easier. The core of the package is a function kendall.many, which automatically runs partial Kendall correlation tests, computing significance and effect sizes for two lists of variables (x × y) controlling for another set of confounding variables. This nonparametric method can analyze any type of variable except non-binary categorical, and is not sensitive to the type of data distribution or the presence of outliers in the data. The calculation is using the ppcor package [1] as a template, but uses package pcaPP (nlog(n) complexity) together with the corpcor package instead of the standard R cov implementation with n^2 complexity. The package also contains functions for exploring many variables together with contingency tables, t-tests, and logistic regressions, which might improve not only data exploration but also data description. Part of the package is also the split.many function that enables running tests automatically, e.g. those implemented in Kendall.many, for any subsets of records in the data frame (e.g. for all subjects, men, women, Rh+ subjects, Rh- subjects, Rh+ men, Rh+ women, Rh- men, and Rh- women), i.e. to split the computation according to the combination of ordinal or categorical variables.

We suggest starting exploration analysis by checking the type of variables using the standard R function str(), then running the whole script containing all necessary functions. If you re-run the script using the same installation of R, you can skip the installation of packages (the first line). Then select the lists of variables for the explorative analysis using selectColumns function (you can use a very large number of variables) and draw a matrix of histograms of these variables for a visual check using printHist.many function. Then you can start exploring your data by searching for associations between variables using the function Kendall.many or, in the case of categorical variables, using the function tables.many. The patterns in the output .csv file can be most conveniently found in Excel using the functions of conditional formatting.

When performing an explorative analysis, it is always necessary to apply a correction for multiple tests, e.g. the Benjamini-Hochberg procedure [2] with a false discovery rate pre-set somewhere between 0.1 and 0.25 to quantify the risk of statistical artifacts. Most importantly, any association discovered in the explorative phase of a study should be confirmed by data obtained in an independent, preferably preregistered, study.

The package Explorer v. 1.0 contains nine functions:

1) **Function selectColumns(d)** for preparing a vector with the names of variables for future processing.

The function opens the window with the names of all variables present in a specified data frame (d), and the user can select or deselect the names for future analyses. The names are in a vector in a proper format (in quotes, separated by comas). The vector of names is also copied into the clipboard.

Example of how to use the function:

x = selectColumns(colnames(d))

2) **Function orderColumns** for moving names in the vector up and down or deleting items in the vector. This function can be used, for example, for changing the order of the variables for the next analysis.

Example of how to use the function:

x = orderColumns(x)

3) **Function printHist.many**(d, x, "histograms.png", 7,) for drawing a matrix of histograms for variables in a vector of variable names. d: data frame, x: list of variables names in the data frame d, " histograms.png": name of the output file, 7: number of histograms in one row. Each histogram also shows the number of cases in the least numerous class, in the most numerous class, and the fraction of all subjects represented by the least numerous class in the whole data set (a useful parameter for binary variables, showing, e.g., the fraction of infected subjects in the whole set).

Example of how to use the function: printHist.many(d, x, " histograms.png", 7,)

4) **Function Kendall.many** that computes partial Kendall correlation tests between binary, ordinal, and continuous variables in list x and list y, controlling for binary, ordinal, and continuous variables in list z. The covariate that is the subject of analysis is removed from list of z for a particular analysis. The function Kendall.many prints nothing – you must use the function (tisk1) to write the results to the .csv file. Alternatively, you can use the function split.many to run the Kendall analyses for subsets of records of the data frame; in this case, the results are written right away into the .csv file, see the description of the function split.many below. The output file (which can be opened in Excel) contains seven tables x × y with partial Kendall Taus, p values, N, N in the rarest class, Pearson r, R-squared, and Cohen's f. If you run tisk1 more times with the same name, the results are appended to the same .csv file.

Example of how to use the function:

either using two commands: results = Kendall.many(d, x, y, z)   tisk1("nameOutputFile",results)
or using one command: Kendall.many(d, x, y,z)$save("nameOutputFile ")
or using the function split.many: split.many(d, s, "splitKendall", funkce = Kendall.many, x=x, y=y, z=z)

5) **Function t.test.many** computes t-tests for lists of binary variables (x) and continuous variables (y) from the dataframe d. Using the function save.t (or internal function save) you can save the results (mean1, mean2, t-value, df, p, Cohen d, N1, N2, std.1, std2, f, fvar, fp) to a .csv file.

Example of how to use the function:

either using two commands: j<-t.test.many(d, y, x)   save.t("output", j)

or using one command   t.test.many(d, y, x)$save("output")

or using the function split.many: split.many(d, s, "splitTest", funkce = t.test.many, x=x, y=y)

6) **Function tables.many** computes contingency tables for lists of binary, ordinal, or categorical variables x and y. The results N1, N1%, N2, N2%, Chi^2, p Chi, Kruskal statistics, p Kruskal can be saved using the function tisk1.

Example of how to use the function:

either using two commands: j<- tables.many(d, y, x)   save.t("output", j)

or using one command

tables.many(d, y, x)$save("output")

or using the function split.many: split.many(d, s, "splitTables", funkce = tables.many, x=x, y=y)

7) **Function tables.bin.many** computes contingency tables and Fisher tests for two lists of binary variables x and y. The results N--, N-+, N+-, N++, N--%, N-+%, N+-%, N++%, OR, CI$_{95}$low, CI$_{95}$high, p can be saved using the function save.conting.

Example of how to use the function:

either using two commands: ddd=tables.bin.many(d,x,x)

save.conting("conting.csv", ddd)

or using one command: save.conting("conting2.csv", tables.bin.many(d,x,x))

8) **Function logist.many** computes N logistic regressions for lists of N dependent binary variables (x) and a set of independent variables (y). The binary variables must be coded 0,1. The results (OR, C.I.95 low, C.I.95 high, p, Estimate, Std error, Z, and d.f.) can be printed to .csv file using, e.g., the function save.logist.

Example of how to use the function:

either using two commands: j=logist.many(d,x,y)   save.logist("logist", j)

or using one command: logist.many(d,x,y)$save("logist2")

or using the function split.many: split.many(d, "sex", "splitLog", funkce = logist.many, x=x, y=y)

9) **Function split.many** enables split computation according to one or more binary, ordinal, or categorial variables listed in the vector s, i.e. run the functions Kendall.many, ancova.many for subsets of records in the data frame (e.g. for all subjects, men, women, Rh+ subjects, Rh- subjects, Rh+ men, Rh+ women, Rh- men, and Rh- women)

Examples of how to use the function:

split.many(d, s, "splitTest", funkce = t.test.many, x=y, y=x)

split.many(d, s, "splitKendall", funkce = Kendall.many, x=x, y=y, z=z)

1 Kim, S. ppcor: An R package for a fast calculation to semi-partial correlation coefficients. *Commun. Stat. Appl. Methods* **22**, 665–674, doi:doi: 10.5351/CSAM.2015.22.6.665 (2015).

2 Benjamini, Y. & Hochberg, Y. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *J. Roy. Stat. Soc. B Met.* **57**, 289-300 (1995).

Apendix

Copy the whole script to R and run it before the analysis.

```
install.packages(c("tcltk2","ppcor","corpcor", "pcaPP", "psych"))

library(corpcor)
library(pcaPP)
library(tcltk2)
library(ppcor)
library(psych)


#########################################################1
selectColumns = function(columns) {
  win1 <- tktoplevel()
  win1$env$lst <- tk2listbox(win1, width=100, height = 15, selectmode = "multiple")
  tkgrid(win1$env$lst, padx = 10, pady = c(5, 10))
  for (column in columns)
    tkinsert(win1$env$lst, "end", column)
  onOK <- function() {
    win1$env$choice <- unlist(columns[as.numeric(tkcurselection(win1$env$lst)) + 1])
    writeClipboard(paste0("\"", win1$env$choice, "\"", collapse = ","))
    tkdestroy(win1)
  }
  win1$env$butOK <-tk2button(win1, text = "OK", width = -6, command = onOK)
  tkgrid(win1$env$butOK, row = 2)
  tkwait.window(win1)
  return(win1$env$choice)
}
#########################################################2
orderColumns = function(columns) {
  win1 <- tktoplevel()
  win1$env$lst <- tk2listbox(win1, width=100, height = 15, selectmode = "single")
  tkgrid(win1$env$lst, padx = 10, pady = c(5, 10))
  for (column in columns)
    tkinsert(win1$env$lst, "end", column)
  tkselection.set(win1$env$lst, 2)
  onOK <- function() {
    tkdestroy(win1)
  }
  win1$env$butOK <-tk2button(win1, text = "OK", width = -6, command = onOK)
  onUP = function() {
    pos = as.numeric(tkcurselection(win1$env$lst))
    if(pos == 0) {
      return()
    }
    text = tkget(win1$env$lst, pos)
    tkdelete(win1$env$lst, pos)
    tkinsert(win1$env$lst, pos-1, text)
    columns[pos+1] = columns[pos]
    columns[pos] = as(text, typeof(columns))
    tkselection.set(win1$env$lst, pos-1)
    win1$env$result = as(tkget(win1$env$lst, 0, length(columns) - 1), typeof(columns))
  }
  win1$env$butUP <-tk2button(win1, text = "up", width = -6, command = onUP)
  onDOWN = function() {
    pos = as.numeric(tkcurselection(win1$env$lst))
    if(pos == length(columns) - 1) {
      return()
    }
    text = tkget(win1$env$lst, pos)
    tkdelete(win1$env$lst, pos)
    tkinsert(win1$env$lst, pos+1, text)
    columns[pos+1] = columns[pos+2]
    columns[pos+2] = as(text, typeof(columns))
    tkselection.set(win1$env$lst, pos+1)
    win1$env$result = as(tkget(win1$env$lst, 0, length(columns) - 1), typeof(columns))
  }
  win1$env$butDOWN <-tk2button(win1, text = "down", width = -6, command = onDOWN)
  onDel = function() {
    pos = as.numeric(tkcurselection(win1$env$lst))
    text = tkget(win1$env$lst, pos)
    tkdelete(win1$env$lst, pos)
    columns = setdiff(columns, as(text, typeof(columns)))
    win1$env$result = as(tkget(win1$env$lst, 0, length(columns) - 1), typeof(columns))
  }
  win1$env$butDel <-tk2button(win1, text = "delete", width = -6, command = onDel)


  tkgrid(win1$env$butOK)
  tkgrid(win1$env$butUP)
  tkgrid(win1$env$butDOWN)
  tkgrid(win1$env$butDel)
  tkwait.window(win1)
  return(win1$env$result)
}

####################################################################################3
printHist.many = function(data, cols,jmeno="histograms.png", colums=5, resol=500) {
  resolution = resol
  num = ceiling(length(cols)/colums)
  png(jmeno, width=colums*resolution,height=num*resolution,res=200)
  par(mar=c(2,5,1,1),mfrow=c(num,colums))
  for (var in cols) {
    print(var)
    if(is.numeric(data[[var]])) {
      hist(data[[var]], main=paste(var,min(table(data[[var]])), max(table(data[[var]])),
round(min(table(data[[var]]))/sum(table(data[[var]])),5), sep = ", "))
    }
    else {
      barplot(table(data[[var]]),
main=paste(var,min(table(data[[var]])),round(min(table(data[[var]]))/sum(table(data[[var]])),5
), sep = ", "), horiz=T, las=1)
    }
  }
  dev.off()
}
####################################################################################4
Kendall.many = function(data, x, y, z = c()) {
  nenumeric = c()
  for(col in c(x,y,z)) {
    if(!is.numeric(data[[col]])) {
      nenumeric = c(nenumeric,col)
    }
  }

  z= setdiff(z,nenumeric)
  y= setdiff(y,nenumeric)
  x= setdiff(x,nenumeric)

  #fill missing z values with mean
  for(z1 in z) {
    data[paste("tmp",z1)] = is.na(data[[z1]])
    data[is.na(data[[z1]]),z1] = mean(unlist(data[z1]), na.rm=T)
  }

  #gather pcor results
  result_pcor = array(dim=c(length(x),length(y), 4), dimnames=list(x, y, c("estimate", "p",
"n", "min (N of subjects in the rarest class)")))
  homogenous = c()
  for(col in c(x,y,z)) {
    if(length(setdiff(unique(data[[col]]), NA)) == 1) {
      homogenous = c(homogenous,col)
    }
  }

  z=setdiff(z,homogenous)

  xpos = 0
  for(x1 in x) {
    ypos = 0
    xpos = xpos + 1
    if(x1 %in% homogenous) next()
    print(paste0("              x: ", x1))
    for(y1 in y) {
      ypos = ypos + 1
      if(y1 %in% homogenous) next()
      print(y1)
      #fill with NA if both variables are the same
      if(x1 == y1) {
        result_pcor[xpos,ypos,]=rep(NA, times=4)
        next()
      }

      z1 = setdiff(z, c(x1,y1))

      #ignore any rows with a missing value
      indices = !is.na(data[x1]) & !is.na(data[y1])
      data1 = data[indices,]

      if(length(data1[[x1]]) < 3) next()

      h=tryCatch({
        unlist(pcor.fk.test(data1[x1], data1[y1], data1[z1])[,c("estimate", "p.value", "n")])
      },
      error=function(err) {
        message(err)
        return(NA)

      })

      h = c(h,min(if(!x1 %in% z) table(data1[[x1]]) else
table(data1[!data1[[paste("tmp",x1)]],x1])))
      result_pcor[xpos,ypos,] = h
    }
  }

  #prepare wanted data
  result = list()
  result$partial_Kendall_Tau = array(result_pcor[,,"estimate",
drop=F],dim=dim(result_pcor)[c(1,2)],dimnames = dimnames(result_pcor)[c(1,2)])
  result$p = array(result_pcor[,,"p", drop=F],dim=dim(result_pcor)[c(1,2)],dimnames =
dimnames(result_pcor)[c(1,2)])
  result$n = array(result_pcor[,,"n", drop=F],dim=dim(result_pcor)[c(1,2)],dimnames =
dimnames(result_pcor)[c(1,2)])
  result[["min (N of subjects in the rarest class)"]] = array(result_pcor[,,"min (N of
subjects in the rarest class)", drop=F],dim=dim(result_pcor)[c(1,2)],dimnames =
dimnames(result_pcor)[c(1,2)])
  result$pearson = sin(result$partial_Kendall_Tau*pi*0.5)
  result$rsquared = result$pearson^2
  result$cohen_f = sqrt(result$rsquared/(1-result$rsquared))
  result$covars = z

  if (length(nenumeric)>0) {
    print("These nonnumeric variables were skipped:")
    print(unique(nenumeric))
  }

  result = list(form=deparse(sys.calls()[[sys.nframe()]]),data=result, save=function(filename,
...) tisk1(filename, result, ...))
  return(result)
}


pcor.fk=function (x)
{
  if (is.data.frame(x))
```

```
    x <- as.matrix(x)
  if (!is.matrix(x))
    stop("supply a matrix-like 'x'")
  if (!(is.numeric(x) || is.logical(x)))
    stop("'x' must be numeric")
  stopifnot(is.atomic(x))
  n <- dim(x)[1]
  gp <- dim(x)[2] - 2
  cor <- pcaPP::cor.fk(x)
  pcor <- corpcor::cor2pcor(cor)
  dimnames(pcor) = dimnames(cor)
  diag(pcor) <- 1
  statistic <- pcor/sqrt(2 * (2 * (n - gp) + 5)/(9 * (n - gp) * (n - 1 - gp)))
  p.value <- 2 * pnorm(-abs(statistic))
  diag(statistic) <- 0
  diag(p.value) <- 0
  list(estimate = pcor, p.value = p.value, statistic = statistic, n = n, gp = gp)
}

pcor.fk.test = function (x, y, z)
{
  x <- c(x)
  y <- c(y)
  z <- as.data.frame(z)
  xyz <- data.frame(x, y, z)
  pcor = pcor.fk(xyz)
  data.frame(estimate = pcor$est[1, 2], p.value = pcor$p.value[1, 2], statistic =
pcor$statistic[1, 2], n = pcor$n, gp = pcor$gp)
}

tisk1 = function(filename, tabulky) {
  write(tabulky$form, filename, append=T)
  tabulky=tabulky$data
  filename = paste0(filename, ".csv")
  write(paste(tabulky$covars, collapse=";"), filename, append=T)
  for(tabulka in names(tabulky[names(tabulky) != "covars"])) {
    write(tabulka, filename, append=T)
    suppressWarnings(write.table(tabulky[[tabulka]], filename, sep = ";", append=T, col.names
= NA))
  }
}

###############################################################################################
##############################5
t.test.many = function(data, x, y) {
  #gather pcor results
  columnlist = c("mean1", "mean2", "t-value", "df", "p", "cohen-d", "n1", "n2", "std.dev1",
"std.dev2", "f", "fvar", "fp")
  result_pcor = array(dim=c(length(x), length(columnlist), length(y)), dimnames=list(x,
columnlist, y))
  xpos = 0
  for(x1 in x) {
    ypos = 0
    xpos = xpos + 1
    print(paste0("              x: ", x1))
    for(y1 in y) {
      ypos = ypos + 1
      print(y1)
      #fill with NA if both variables are the same
      if(x1 == y1) {
        result_pcor[xpos,ypos,]=rep(NA, times=length(y))
        next()
      }
      #ignore any rows with a missing value
      indices = !is.na(data[x1]) & !is.na(data[y1])
      data1 = data[indices,]
      if(length(data1[[x1]]) < 3) next()
      h=tryCatch({
        ttest=t.test(data[[x1]] ~ data[[y1]])
        summ = describeBy(data[x1], data[y1], mat=T)
        splitcol = split(data[[x1]], data[[y1]])
        ftest = var.test(splitcol[[1]], splitcol[[2]])
        cohen = cohen.d(data[c(x1,y1)], y1)
        c(summ[1, "mean"], summ[2, "mean"], ttest[["statistic"]], ttest[["parameter"]],
ttest[["p.value"]], cohen$cohen.d[2], summ[1, "n"], summ[2, "n"], summ[1, "sd"], summ[2,
"sd"], ftest[["statistic"]], ftest[["estimate"]], ftest[["p.value"]])
      },
      error=function(err) {
        message(err)
        return(NA)
      })
      result_pcor[xpos,,ypos] = h
    }
  }
  result = list(form=deparse(sys.calls()[[sys.nframe()]]),data=result_pcor,
save=function(filename, ...) save.t(filename, result, ...))
  return(result)

}



# Ukládaní výsledků do Excelu
save.t = function(filename, tabulky) {
  filename = paste0(filename, ".csv")
  write(tabulky$form, filename, append=T)
  tabulky = tabulky$data
  for(tabulka in dimnames(tabulky)[[3]]) {
    write(tabulka, filename, append=T)
    tmp = array(dim = dim(tabulky)[c(1,2)],dimnames = dimnames(tabulky)[c(1,2)])
    tmp[,] = tabulky[,,tabulka]
    suppressWarnings(write.table(tmp[,, drop=F], filename, sep=";", append=T, col.names=NA))
    write("", filename, append=T)
  }
}
################################################################################################
#############################6
interleave = function(a) {
  n = dim(a)[2]/2
  s = rep(1:n, each = 2) + (0:1) * n
  return(a[,s])
}
```

```
tables.many= function(data, x, y) {
  result = list()
  for(x1 in x) {
    for(y1 in y) {
      ta = table(data[c(x1,y1)])
      p = prop.table(ta,margin=1)*100
      result[[paste(x1,y1)]] = list(interleave(cbind(ta,
p)),chisq.test(ta)[1:3],kruskal.test(data[[x1]],data[[y1]])[1:3])
    }
  }
  result = list(form=deparse(sys.calls()[[sys.nframe()]]),data=result, save=function(filename,
...) tisk1(filename, result, ...))
  return(result)

}

###############################################################################################
############################7
tables.bin.many= function(data, x, y) {
  columnlist = c("","","--","-+","+-","++","--%","-+%","+-%","++%","OR","low","high","p")
  result = array(dim=c(length(x)*length(y), length(columnlist)),
dimnames=list(c(t((outer(x,y,paste,sep=",")))), columnlist))
  for(x1 in x) {
    for(y1 in y) {
      cont=table(data[[x1]], data[[y1]],dnn = c(x1,y1))
      prop=prop.table(cont,margin=1)*100
      colnames(cont) = rownames(cont) = c(1,2)
      fis=fisher.test(cont)
      result[paste(x1,y1,sep=","),1] = x1
      result[paste(x1,y1,sep=","),2] = y1
      result[paste(x1,y1,sep=","),3] = cont[1,1]
      result[paste(x1,y1,sep=","),4] = cont[1,2]
      result[paste(x1,y1,sep=","),5] = cont[2,1]
      result[paste(x1,y1,sep=","),6] = cont[2,2]
      result[paste(x1,y1,sep=","),7] = prop[1,1]
      result[paste(x1,y1,sep=","),8] = prop[1,2]
      result[paste(x1,y1,sep=","),9] = prop[2,1]
      result[paste(x1,y1,sep=","),10] = prop[2,2]
      result[paste(x1,y1,sep=","),11] = fis$estimate
      result[paste(x1,y1,sep=","),c(12,13)] = fis$conf.int
      result[paste(x1,y1,sep=","),14] = fis$p.value
    }
  }
  result = list(form=deparse(sys.calls()[[sys.nframe()]]),data=result, save=function(filename,
...) save.conting(filename, result, ...))
  return(result)
}



save.conting=function(name,table){
  write.table(table$data, file=paste0(name, ".csv"), row.names = F, sep = ";", append = T)
}


###############################################################################################
##############################8
```

```
logist.many = function(data, x, y) {
  #gather epidisplay results
  columnlist = c("OR", "ORlo", "ORhigh", "p",  "Estimate", "Std.Error", "z.value","degrees")
  result_pcor = array(dim=c(length(x), length(columnlist),length(y)), dimnames=list(x,
columnlist, y))
  factorform = paste(y,collapse="+")
  for(x1 in x) {
    tryCatch({
      form = as.formula(paste(x1," ~ ", factorform))
      model = glm(form,data=data,family = binomial(link="logit"))
      summ = summary(model)
      result_pcor[x1,"OR",] = exp(coef(model)[-1])
      result_pcor[x1,"ORlo",] = exp(confint(model)[-1,1])
      result_pcor[x1,"ORhigh",] = exp(confint(model)[-1,2])
      result_pcor[x1,"p",] = summ$coefficients[-1,4]
      result_pcor[x1,"Estimate",] = coef(model)[-1]
      result_pcor[x1,"Std.Error",] = summ$coefficients[-1,2]
      result_pcor[x1,"z.value",] = summ$coefficients[-1,3]
      result_pcor[x1,"degrees",] = summ$df.null
    },
    error=function(err) {
      message(err)
    })
  }
  result = list(form=deparse(sys.calls()[[sys.nframe()]]),data=result_pcor,
save=function(filename, ...) save.logist(filename, result, ...))
  return(result)

}

# Ukládaní výsledků do Excelu
save.logist = function(filename, tabulky) {
  filename = paste0(filename, ".csv")
  for(tabulka in dimnames(tabulky$data)[[3]]) {
    write(tabulka, filename, append=T)
    tmp = array(dim = dim(tabulky$data)[c(1,2)],dimnames = dimnames(tabulky$data)[c(1,2)])
    tmp[,] = tabulky$data[,,tabulka]
    suppressWarnings(write.table(tmp[,, drop=F], filename, sep=";", append=T, col.names=NA))
    write("", filename, append=T)
  }
}

#############################################################################################
############################9
split.many = function(data, sorters, output="manyVys", all=T, funkce = Kendall.many, ...) {
  vysledek = funkce(data, ...)
  while(file.exists(paste0(output,".csv"))) output=paste0(output, "1")
  write("All groups", paste0(output,".csv"), append = T)
  vysledek$save(output)

  variants = list()
  for(i in 1:length(sorters)) {
    variants = c(variants, combn(sorters, i, simplify = F))
  }
  print(variants)
  for(variant in variants) {
    groups = list(data)
    levels = list("")
    for(sorter in variant) {
      levly = setdiff(unique(data[[sorter]]), NA)
      newgroups = list()
      newlevels = list()
      i = 0
      for(group in groups) {
        i = i + 1
        newgroups[[i]] = lapply(levly, function(levl)  {
          group[group[[sorter]] %in% levl,]
        })
        newlevels[[i]] = lapply(levly, function(levl) {
          paste0(levels[(i-1)%%length(levly) + 1], sorter,": ", levl, ";")
        })
      }

      levels = unlist(newlevels, F)
      groups = unlist(newgroups, F)
    }

    for(i in 1:length(groups)) {
      levl = levels[[i]]
      group = groups[[i]]
      vysledek = funkce(group, ...)
      write(paste0(levl, "
*************************************************************************"),
paste0(output,".csv"), append = T)
      vysledek$save(output)
    }
  }
}
```