Tiago A. de Souza

Biólogo (**UNICAMP**), mestre em Genética e Biologia Molecular (**UNICAMP-CNPEM/LNBio**) e doutorando em Biotecnologia (**USP**). O título da minha tese, sob orientação do Prof. **Carlos Menck**, é: "*Análise das alterações genéticas em exomas de camundongos*".
GoogleScholar Lattes
tiagoantonio@gmail.com
Meu principal interesse é entender as ligações entre genótipo e fenótipo e entender os padrões de mutações pontuais nessas ligações, utilizando NGS e o desenvolvimento de novas ferramentas de bioinformática (Perl e R).
Fav tools: Sublime, Trello, GitHub, BitBucket, Galaxy

Meus Exercícios

Página de exercícios resolvidos: exec

Proposta de Trabalho Final

> Oi Tiago,
>
> Sou Gustavo, o monitor encargado de revisar as suas propostas. Em geral achei muito bacana a primeira, só gostaria que explicitasse melhor a estrutura dos objetos do input, e a estrutura dos parâmetros que menciona na segunda parte dessa proposta. Acho que você tem razoavelmente claro o que quer da função, e seria legal ver se consegue escrever ela em formato de pseudocodigo para ver como planeja implementá-la.
>
> A segunda proposta precisa de trabalho, gostaria ver o mesmo nível de elaboração da primeira, lembre-se que as duas fazem parte da avaliação e a segunda não é opcional.
>
> Pode me encontrar no email
>
> Abs,
>
> Gustavo A. Ballen

> Olá Gustavo, tudo bem?

Muito obrigado pelas observações! Inclui as explicações que você pediu sobre os inputs, parâmetros e estrutura da função (plano A e plano B) e também um esquema gráfico de rascunho do output do plano B (que seria o principal desafio dessa função). Tentei não incluir informações específicas da aplicação das funções e tentei focar mais na estrutura e na proposta de cada uma das funções.

Abraços, Tiago

Plano A

A proposta consiste basicamente em criar uma função que receba um arquivo .csv de formato consistente contendo variáveis (10+7*n) associadas a n replicatas de n1 condições experimentais. Cada linha corresponde a uma observação associada a uma determinada variável identificadora. O dataframe possui um formato bem definido - o numero de colunas depende somente do numero n de replicatas e de condições experimentais. A primeira etapa consiste no calculo da média de uma das variaveis para cada observação (n replicatas) de cada condição experimental. A segunda etapa consiste no cálculo da razão log2 entre a média de todas as combinações doias a dois das (n1!/(n1-2)!)/2 condições experimentais. A função deve retornar um plot da razao para cada combinação em que observações >0 e <0 e devem estar destacadas. A função também deve retornar um dataframe contendo a razao da média de todas as combinações 2 a 2 das condições experimentais, contendo algumas variaveis do dataframe original para possibilitar a identificacao de cada razao.

A função poderá também, através de um parâmetro especifico, possibilitar uma analise grafica exploratória dos dados, que consistirá da análise de boxplot e histogramas associados a pelo menos duas variáveis de cada observação e grupo experimental. Essas variáveis não afetam diretamente o calculo da razao anterior mas deverão possibilitar a escolha do usuário (através de um ou mais parâmetros - entre 3 ou 4) em utilizar critérios para o calculo da razão através desses parâmetros numéricos.

Portanto, a função deverá se comportar, dependendo dos parâmetros iniciais - como uma função de comparação e cálculo entre condições experimentais ou como uma função exploratória de variáveis controle para o cálculo comparativo principal. Além disso, o mais importante: o usuário poderá filtrar os dados para o cálculo principal usando os parâmetros mais adequados - de acordo com a analise exploratoria dada pela propria funcao.

Estrutura do input: Arquivo em formato tabular; tabela com 7 variáveis (colunas) fixas e um número N x (número de replicatas) de colunas com dados numéricos de intensidade, que correspondem a amostras (cujo número é variável). Após esse número N de colunas estarão grupos de 4 colunas correspondentes a cada amostra, em sequência, que seriam alvo dos parâmetros pelo usuário. Basicamente os dados correspondem a dados de proteômica, cujas linhas (observações) correspondem a proteínas. Esse arquivo é gerado diretamente por pipelines de espectrometria de massas.

Parâmetros: Usar as informações do grupo das 4 colunas finais, como por exemplo o número de peptídeos identificados e valores de score, para filtrar somente observações que estejam nesse critério. Ex: número de peptídeos identificados > 2.

Estrutura da Função Checagem de parametros (n de amostras>1...). Leitura do .csv → data table. Matriz somente com os dados correspondentes aos valores de intensidade das amostras. Se analise exploratoria > graficos de distribuição dos valores de “qualidade” - histogramas e boxplots. Se analise comparativa > Recebe parametros de controle (pep, score) criando um subset do dataframe inicial recebido. Calcula a média das replicatas de cada amostra da matriz de acordo com o numero de amostras e numero de replicas (parametros fornecidos pelo usuario). Calcula a razão de todas as combinações 2 a 2 das amostras, log2. Cria graficos de dispersão de todas as combinações 2 a 2. Utiliza um parametro fornecido pelo usuário de um identificador de uma proteina housekeeping. Normalização da media das intensidades pelo valor de intensidade da proteina housekeeping. Criação de um heatmap com os valores de media de intensidade normalizados. Retorna ao usuário as médias de intensidade de cada amostra e também uma matriz com os valores de razao de cada combinação. Além dos gráficos.

> Oi Tiago, Adorei o seu fluxograma. Só não entendi por que você precisa necessariamente de 7 colunas e das 4 colunas finais. Isto não é específico demais? Tente pensar em estruturas de dados diferentes das suas, por exemplo, e como a sua função lidaria com isto. Talvez especificar quais colunas contem quais informações seja uma solução mais adequada do que engessar o número de colunas. — *Sara Mortara*

> Olá Sara, tudo bem? Muito obrigado pelas suas observações! Na verdade essa estrutura da tabela é a estrutura geral padronizada desse tipo de arquivo, ou seja, simplificadamente é a saída de vários outros programas. Portanto, a função lidaria com a maior parte das saídas de experimentos de proteoma, independentemente da quantidade ou tipo de amostras do usuário (o usuario fornecerá esses dados através de parâmetros). Eu tentei descrever mais ou menos a estrutura geral do arquivo de entrada e como a função lidaria com variações das tabelas de entrada em relação à localização das colunas. (não sei se fui claro...) Ah, o fluxograma é legal mesmo, e bem tranquilo de usar. Eu uso esse (https://www.lucidchart.com/), é free! — *Tiago*

> GAB: Oi Tiago e Sara,

> Uma alternativa para fazer fluxogramas diretamente no R é o pacote DiagrammeR. Usei na minha tese e achei legal.

Plano B

A proposta B consiste em receber um ou mais dataframes cujas variáveis incluem strings de identificação, agrupamento e localização, e uma variável numérica-alvo. A tarefa primária da função seria ranquear as observações segundo critérios fornecidos pelos usuários através de parâmetros indicativos, por exemplo, quantis ou porcentagem acima da média ou mediana. A função secundária seria criar uma saída gráfica para esse ranqueamento, que consistiria em esquemas gráficos de localização relativa em cada agrupamento em cada dataframe de entrada, contendo as informações de identificação das observações ranqueadas.

A saída gráfica seria o maior desafio, já que consiste em um gráfico não usual, criado a partir de segmentos, linhas e texto e não através de uma função gráfica usual. Basicamente seria como a indicação da localização relativa de hotspots de mutações em cromossomos de virtualmente qualquer organismo.

Estrutura do Input: Dois ou mais arquivos .csv cujas variáveis correspondem a identificadores de localização cromossômica e uma variável correpondente a um numero X de mutações encontradas em um intervalo Z. Essa tabela pode ser gerada por diferentes tipos de estratégia. Cada arquivo é correspondente a uma amostra.

Parâmetros: Além dos arquivos, o usuário deverá fornecer critérios para o ranqueamento das regiões com maior número de mutações como a média ou a mediana e também um valor numérico correspondente a qual será o valor de corte (em relação à media ou a mediana) para a seleção de hotspots, por exemplo em termos de quantis ou valores acima da média.

Estrutura da função: Receber cada arquivo em diferentes data-frames. Calcular valores de média e mediana para cada um deles. Criar subsets de acordo com os parâmetros fornecidos. Criar dois tipos de saídas gráficas para os subsets filtrados contendo também o nome do gene em que o hotspot está inserido. Gráfico 1 → Estruturado em um círculo único com a posição relativa dos hospots encontrados (com cores diferentes para cada amostra) e “espinhos” de tamanho proporcional ao número de mutações encontradas e o nome do gene em que o hotspot está inserido (basicamente uma mistura de segmentos de reta e poligonos). Grafico 2 → Um conjunto de n segmentos (correspondentes ao numero de cromossomos) com segmentos verticais de tamanho proporcional ao numero de mutações e com cores diferentes para cada amostra.

Saídas graficas (output): Grafico 1 - Esquema tipo “mamona” ou “mina terrestre” dos hotspots. Grafico 2 - Esquema mais simples, em que cada segmento é um cromossomo.

> Oi Tiago, eu particularmente adorei a ideia da função fazer este gráfico. Acho um desafio muito interessante a ser implementado. Pense com carinho nas duas propostas e

escolha a que te motive mais! — *Sara Mortara*

Oi Sara, que legal que você gostou! Na verdade o plano A já está bastante adiantado mas minha idéia é garantir o plano A e fazer o plano B também! — *Tiago*

GAB: Ótimo Tiago, go for it! Me diga se precisar de alguma outra coisa, acho que você esta bem no caminho da implementação.

Página de Ajuda

Olá Sara e Gustavo, Consegui terminar o plano B e realmente ele se encaixou melhor na proposta do trabalho final do que o plano A e portanto, se não tiver nenhuma problema podem considerá-lo para a avaliação. De qualquer forma, o plano A estava bem adiantado e então coloquei ele no final da página também (a função já está sendo bem útil!). — *Tiago*

Plano B - flufflyball.plot

```
flufflyball.plot    R Documentation

Plot beautiful sun graphs like a fluffly ball.

Description:

Produce a stylized sun-like plot of the given (grouped) values and returns
unique id names for line values plotted. The length of the line corresponds
to normalized values sweeped around clockwise according to coordinates
provided by user.

Usage:

flufflyball.plot (id,group,coord,values,...)

Arguments:

id
```

```
A character vector where ids corresponds to the id name of each observation.
group
A numeric or integer vector where group corresponds to the group of each
observation
coord
A numeric or integer vector where coord corresponds to coordinates of each
observation
values
A numeric or integer vector where values corresponds to the values of each
observation to be represented in the plot
filter
A natural number for a threshold of the minimal value to be represented in
the plot. Default: filter=0
cex.circle
A numerical value giving the amount by which the centered circle should be
magnified for . Default:cex.circle=6
plot.legend
Logical. If TRUE plots a legend. Default:plot.legend=TRUE
cex.legend
A numerical value giving the amount by which the legend should be magnified
for . Default:cex.legend=6
exclude.na
Logical. If TRUE exclude all observations with any NA.
Default:exclude.na=TRUE

Value:

Returns a vector for unique "ids" ("id" levels) filtered by parameter
'filter' and plotted.

Author(s):

Tiago A. de Souza (tiagoantonio@gmail.com) & github.com/tiagoantonio/

Examples:


flufflyball.plot(id=sample(LETTERS,5000,replace = TRUE),
group=sample(1:4,5000, replace=TRUE), coord=sample(1:1000,5000,
replace=TRUE), values=sample (0:100,5000, replace=TRUE))

flufflyball.plot(id=sample(letters,2000,replace = TRUE),
group=sample(1:4,2000, replace=TRUE), coord=seq(1:2000), values=seq(1:2000),
cex.circle = 9, cex.legend=1.1)

flufflyball.plot(id=sample(LETTERS,50,replace = TRUE), group=rep(1,50),
coord=sample(1:100,50), values=rnorm(50,mean=100,sd=50), cex.circle = 20,
cex.legend = 1.1)

flufflyball.plot(id=sample(letters,5000,replace = TRUE),
```

```
group=sample(1:4,5000, replace=TRUE), coord=sample(1:1000,5000,
replace=TRUE), values=sample (0:100,5000, replace=TRUE), cex.circle=15,
plot.legend=FALSE, filter=80)

flufflyball.plot(id=sample(LETTERS,1000,replace = TRUE),
group=sample(1:2,1000, replace=TRUE), coord=seq(1:1000),
values=sort(rnorm(1000,mean=100,sd=7), decreasing = TRUE))



## The function is currently defined as
fluffyball.plot=function(id,group,coord,values, filter=0, plot.legend=FALSE,
cex.circle=6, cex.legend=0.5, exclude.na=TRUE)
```

Função fluffly.ball.plot

```
################################################################################
###
# May,12 2016
# flufflyball.plot
#
# Plot beautiful sun graphs like a fluffly ball.
#
# Help: flufflyball.plot.Rd
#
# Author: Tiago A. de Souza
# tiagoantonio@gmail.com
# github.com/tiagoantonio/
#
################################################################################
####



flufflyball.plot=function(id,group,coord,values, filter=0, cex.circle=6,
plot.legend=TRUE, cex.legend=0.5, exclude.na=TRUE){
  # main parameter checking
  if(length(id)!=length(group)||length(group)!=length(values)){stop("vectors
must be the same lengths")} # main length checking
  id=as.character(id) #receiving vector id and coerces into character.
  if(class(group)!="integer" & class(group)!="numeric" ){ # checks if group
is integer or numeric
    stop("groups is not a vector of integers")}  # stop with warning msg
  if(class(coord)!="integer" & class(group)!="numeric"){ #checks if coord is
interger or numeric
    stop("coord is not a vector of integers")} # stop with warning msg
  if(class(values)!="numeric" & class(values)!="integer" ){ # checks if
values is numeric
    stop("values is not a vector of numbers")} # stop with warning msg
  if(class(filter)!="numeric"){ # checks if filter is numeric
    stop("values is not numeric")} # stop with warning msg
  if(filter<0){ #checks if filter is >0
    stop("filter for values must be >0")} # stop with warning msg
```

```
  if(cex.circle<0) { #checkis if cex.circle is >0
    stop("cex.circle must be >0")} # stop with warning msg
  # parameter exclude .na for NAs in values
  if (exclude.na==TRUE){ # if exclude.na=TRUE
    group=group[!is.na(values)] #remove elements in group if there is NA is
values
    coord=coord[!is.na(values)] #remove elements in coords if there is NA is
values
    values.checked=values[!is.na(values)] # object values.checked without
NAs (passed)
  }
  if (exclude.na==FALSE){ #if exclude.na=FALSE
    if(any(is.na(group))){stop("There are NAs in group. Please review data
or use exclude.na=TRUE")}  # stop with warning msg
    if(any(is.na(coord))){stop("There are NAs in coord. Please review data
or use exclude.na=TRUE")} # stop with warning msg
    if(any(is.na(values))){stop("There are NAs in values. Please review data
or use exclude.na=TRUE")} # stop with warning msg
    values.checked=values # object values.checked without NAs (passed)
  }
  # different groups
  ngroup=factor(group) # coerces group to factor
  ngroup=length(levels(ngroup)) # total number of levels in object ngroup
  # grid for graphics based on total number of different groups
  if (ngroup==0){stop("group must be different from 0")} # stop with warning
msg if ngroup==0
  if (ngroup==1){par(mfrow=c(1,1))} #divides 1x1 if ngroup=1
  if (ngroup==2){par(mfrow=c(2,1))} #divides 2x1 if ngroup=2
  if (ngroup>2 & ngroup<=4 ){par(mfrow=c(2,2))} #divides 2x2 if ngroup=2..4
  if (ngroup>4 & ngroup<9 ){par(mfrow=c(3,3))} #divides 3x3 if ngroup=4..9
  if (ngroup>=10 & ngroup<=16 ){par(mfrow=c(4,4))} #divides 4x4 if
ngroup=10..16
  if (ngroup>17 & ngroup<=25 ){par(mfrow=c(5,5))} # divides 5x5 if
ngroup=17..25
  if (ngroup>26){par(mfrow=c(8,8))} # divides 8x8 if ngroup=25..64
  if (ngroup>64){stop("group number must be less than 64")} # stop with
warning msg if ngroup>64
  filtered.names=c() # creates an empty object to save object 'id' levels
filtered by parameter filter
  # loop for fluffy ball plot
  for (i in min(group):max(group)){ # loop to build a plot for each group
    subset.group=group[group==i] # subset of group data
    max.coord=max(coord[group==i]) # max coord value
    norm.angles=(2*pi*coord[group==i])/max.coord # normalized angles for
each coord based on max coord value
    max.value=max(values.checked) # max value for 'values'
    par(mar=c(0,0,2,0)) # setting margins for each plot
    plot(0,xlim=c(-max.value,max.value), # empty plot.
         ylim=c(-max.value,max.value), # lims are defined by max and min
values.
```

```
        axes=F, type="n", main=i) # without axes centered on 0,0. group is
the main title.
    for (ii in 1:length(norm.angles)){ # inside loop to build each spine in
the plot
      radius=values.checked[group==i][ii] # pick a single value "ii" for the
group "i"
      #picking colors according to quartiles
if(values.checked[group==i][ii]>=quantile(values.checked[values.checked!=0])
[1] & # if value "ii" is bigger than 1st quartile
values.checked[group==i][ii]<quantile(values.checked[values.checked!=0])[2])
{color.line="#984EA3"} # AND value "ii" is smaller than 2nd quartile
if(values.checked[group==i][ii]>=quantile(values.checked[values.checked!=0])
[2] & # if value "ii" is bigger than 2nd quartile
values.checked[group==i][ii]<quantile(values.checked[values.checked!=0])[3])
{color.line="#4DAF4A" } # AND value "ii" is smaller than 3rd quartile
if(values.checked[group==i][ii]>=quantile(values.checked[values.checked!=0])
[3] & # if value "ii" is bigger than 3rd quartile
values.checked[group==i][ii]<quantile(values.checked[values.checked!=0])[4])
{color.line="#377EB8"} # AND value "ii" is smaller than 4th quartile
if(values.checked[group==i][ii]>=quantile(values.checked[values.checked!=0])
[4] & # if value "ii" is bigger than 4th quartile
values.checked[group==i][ii]<=quantile(values.checked[values.checked!=0])[5]
){color.line="#E41A1C"} # AND value "ii" is smaller than 5th quartile
      #applying filter parameter and drawing lines
      if (radius>filter){ #if value>filter value(radius)
lines(c(0,sin(norm.angles[ii])*radius),c(0,cos(norm.angles[ii])*radius),
col=color.line) #draw the line using picked color and normalized angles
        filtered.names=append(filtered.names,
id[group==i][ii],after=length(filtered.names)) # adding id to filtered
values in object "filtered names"
      }
    }
    points(0,0,pch=21,cex=cex.circle, bg="White", col=1) # draw a circle in
front of lines using cex.circle user parameter
    if (plot.legend==TRUE){
      legend("bottom", c("4th quartile", "3rd quartile", "2nd quartile",
"1st quartile"), # legend of 4 quartiles
             fill=c("#E41A1C", "#377EB8","#4DAF4A","#984EA3"),  # ColorBrewer
colors
             cex=cex.legend, horiz=TRUE,inset=.02, bg="White", box.lty=0, #
horizontal legend without border
             title=paste(deparse(substitute(values)))) # legend title with
vector group name
    }
  }
  filtered.names=as.factor(filtered.names) # coercing filtered names vector
as factor
  filtered.names=levels(filtered.names) # unique "id" names
  par(mfrow=c(1,1)) # resets par(mfrow)
  return(filtered.names) # return unique filtered "id"
}
```

Download:

Conjunto de dados reais de precipitação e temperatura mínima na cidade de São Paulo (2005-2015) que podem exemplificar melhor a utilidade da função (mais informações em info):
data.chuva.sp.tar.gz

Plano A - hikaku.itrap

Download arquivos de exemplo: exemplo1.csvexemplo2.csv

```
hikaku.itrap    R Documentation
Process label-free comparative proteomics data

Description

This function returns comparative log2 ratio analysis and mean peak area
heatmap comparison from label-free proteomics or an exploratory graphical
sample data analysis.

Usage

hikaku.itrap("data.csv", samplen, nrep, ...)
Arguments

data.csv
A tab-delimited .csv file in Protein Discoverer (ThermoLife) output format.
samplen
A natural number >1 and <16 for samples/conditions
nrep
A natural number >0 for replicates of each sample/condition
expl
TRUE for exploratory analysis and FALSE for comparative label-free
proteomics analysis. Default: expl=FALSE
pep
A natural number for a threshold of the minimal number of peptides detected.
Default: pep=0
psm
A natural number for a threshold of minimal PSM value. Default:psm=0
heat.qt
A rational number (0<x<1) which means the fraction of upper quantile to not
appear in the heatmap. Default:heat.qt=0.1
pnames
TRUE for protein id as names and FALSE for Accession number.
Default:pnames=TRUE
housek
Accession number with "" for the housekeeping protein selected for internal
sample normalization
Author(s)

Tiago A. de Souza (tiagoantonio@gmail.com)
```

```
Examples:

hikaku.itrap("exemplo1.csv",6,3,expl=TRUE)
hikaku.itrap("exemplo1.csv",6,3,expl=FALSE)
hikaku.itrap("exemplo1.csv",6,3,expl=FALSE,pep = 3,pnames = FALSE)
hikaku.itrap("exemplo1.csv",6,3,expl=FALSE,pep = 3,pnames = FALSE,
housek="57283874")
hikaku.itrap("exemplo1.csv",6,3,expl=FALSE,pep = 3,pnames = FALSE,
housek="57283874",heat.qt = 0.01)


hikaku.itrap("exemplo2.csv",2,6,expl=TRUE)
hikaku.itrap("exemplo2.csv",2,6,expl=FALSE)
hikaku.itrap("exemplo2.csv",2,6,expl=FALSE,pep = 3,pnames = FALSE)
hikaku.itrap("exemplo2.csv",2,6,expl=FALSE,pep = 3,pnames = FALSE,
housek="Q2T9S5")
hikaku.itrap("exemplo2.csv",2,6,expl=FALSE,pep = 3,pnames = FALSE,
housek="Q2T9S5",heat.qt = 0.01)
```

Código

```
#############################################################################
###
#
# April, 26 2016
#
# hikaku.itrap
#
# R function for processing label-free comparative proteomics data.
#
#
# Help: hikaku.itrap.Rd
#
# Required packages: gplots, RColorBrewer, numbers
#
# If you have any question or suggestions please contact the author:
#
# Author: Tiago A. de Souza
# tiagoantonio@gmail.com
# github.com/tiagoantonio
#
#
#############################################################################
#####

hikaku.itrap= function(filename="filename",samplen, nrep, expl=FALSE, pep=0,
psm=0, heat.qt=0.1, pnames=TRUE, housek){

#####libraries#####
  if (!require("gplots")) { #package required for heatmap graph
    install.packages("gplots", dependencies = TRUE) #install gplots if
```

```
package is not present
    library(gplots) #load library
  }

  if (!require("RColorBrewer")) { #package required for heatmap
    install.packages("RColorBrewer", dependencies = TRUE) #install
RColorBrewer if package is not present
    library(RColorBrewer) #load library
  }

  if (!require("numbers")) { #package required for prime calculations
    install.packages("RColorBrewer", dependencies = TRUE) #install numbers
if package is not present
    library(RColorBrewer) #load library
  }
####acessory functions######
  PrimeSumRec=function(x){ # grid for graphs
    if (x==2 | x==1){ # if n combinations = 1 or 2
      x=c(2,1)
      return(x)
    }
    if(isPrime(x)==TRUE){ # if prime sum 1 then factorize
      x=x+1
      x=primeFactors(x)
    }
    else {
      x=primeFactors(x) # if not prime factorize
    }
    if (length(x)==2){ # if prime factors number =2 return vector
      return(x)
    }
    if (length(x)==3){ # if prime factors number equal =3
      x1=length(x)-1
      x.sum=x[length(x)]*x[x1]
      x=x[-length(x)]
      x=x[-(x1)]
      x=append(x, x.sum, after = length(x))
    }
    else { # multiply borders of prime factors
      i=2
      x1=i+1
      x.sum=x[i]*x[i+1]
      x=x[-i]
      x=x[-(x1)]
      x=append(x, x.sum)
      x1=length(x)-1
      x.sum=x[length(x)]*x[x1]
      x=x[-length(x)]
      x=x[-(x1)]
      x=append(x, x.sum, after = length(x))
```

```
    }
    return(x) # return vector
  }
  count.zero=function(x){ #function to count the sum of replicates without
0s of conditions/samples
    sum(x!=0)
  }

#### parameter check ######
  if (samplen==1| samplen>=16){
    stop("Please use samplen (number of conditions) > 1 and < 16")
  }
#######csv#########
  report=read.csv(filename, sep="\t", dec = ".", header=T, as.is=T) #
reading main input table tab-delimited csv with dec=,
####parsing areas, nsample and nrep #######
  area.samples=report[8:(7+(samplen*nrep))] #separating sample area data
  rep.matrix=matrix(NA, ncol = samplen, nrow=length(area.samples[,1]), byrow
= F) #matrix NA for sample area data
  i.sample=seq(from=1,to=(samplen*nrep), by=nrep) #counter for number of
samples

###explo analysis##
  if (expl==TRUE){ #expl=TRUE for exploratory analysis
    x11() #first window for pep, psm, score and hist of pep abundance
    par(mfrow=c(2,2)) #4x4 graph window
    pep.column=seq(from=(10+(samplen*nrep)),
to=(6+(samplen*nrep))+((samplen*nrep)*4), by=4) #indexes of peptide columns
for each sample
    boxplot(report[pep.column], outline=F, col=brewer.pal(name="Set3",
n=12), main="Peptides") #boxplot for peptides
    psm.column=seq(from=(11+(samplen*nrep)),
to=(7+(samplen*nrep))+((samplen*nrep)*4), by=4) #indexes of PSM columns for
each sample
    boxplot(report[psm.column], outline=F, col=brewer.pal(name="Set3",
n=12), main="PSM") #boxplot for psm
    score.column=seq(from=(8+(samplen*nrep)),
to=(5+(samplen*nrep))+((samplen*nrep)*4), by=4) #indexes of score columns
for each sample
    boxplot(report[score.column], outline=F, col=brewer.pal(name="Set3",
n=12), main="Score") #boxplot for scores
    hist(report[5][report[5]!=0],xlab="", main="Unique Peptides") #histogram
for unique peptides
    par(mfrow=c(1,1)) #reseting par
    # number of proteins which appears in 0, 1, 2 or 3 replicas
    ii=0 # ii for number of conditions
    for (i in i.sample){ #loop for absence of protein in replicates
      ii=ii+1
      rep.sample=apply(area.samples[i:(i+2)],1,count.zero)#1.2omiting NAs,
NA=0 if NA mean with others
      rep.matrix[,ii]=rep.sample #matrix containing reps from samples
```

```
    }
    x11() # second graph for proteins which appears in replicas
    # checking number of samples to build panel
    if (samplen<=4){
      par(mfrow=c(2,2))
    }
    if (samplen<=6 & samplen>4){
      par(mfrow=c(2,3))
    }
    if (samplen<=9 & samplen>6){
      par(mfrow=c(3,3))
    }
    if(samplen>9) {
      par(mfrow=c(5,5))
    }
    for (i in 1:samplen){ #hist graph for each condition
      histname=paste("Sample",i)
      sum0=sum(rep.matrix[,i]==0) # sum of presence of each protein in
conditions
      sum1=sum(rep.matrix[,i]==1)
      sum2=sum(rep.matrix[,i]==2)
      sum3=sum(rep.matrix[,i]==3)
      hist(rep.matrix[,i], breaks = 6,xlab="", main=histname, xaxt='n')
      axis(1,at=c(0.25,0.75,1.75,2.75), labels=c(0,1,2,3))
      text(x = 0.25,y=30, sum0) # text in histograms
      text(x = 0.75,y=30, sum1)
      text(x = 1.75,y=30, sum2)
      text(x = 2.75,y=30, sum3)
    }
    par(mfrow=c(1,1))
    return() # explo analysis ends here with no output.
  }
  ###Comparative Proteomics Analysis#######
  else{
    #receiving parameters for peptide and psm filtering
    #peptide
    pep.column=seq(from=(10+(samplen*nrep)),
to=(6+(samplen*nrep))+((samplen*nrep)*4), by=4)
    ii=7 #start column
    for(i in pep.column){
      ii=ii+1
      pep1=report[,i]
      pep1[pep1<pep]=NA
      report[,i]=pep1
      report[,ii][is.na(report[,i])]=0
    }
    #psm
    psm.column=seq(from=(11+(samplen*nrep)),
to=(7+(samplen*nrep))+((samplen*nrep)*4), by=4)
    ii=7 # start column
```

```
    for(i in psm.column){
      ii=ii+1
      psm1=report[,i]
      psm1[psm1<psm]=NA
      report[,i]=psm1
      report[,ii][is.na(report[,i])]=0
    }
    ### Checking NAs in area.samples ####
    #areas=0 -> NA
    area.samples=report[8:(7+(samplen*nrep))]
    area.samples[area.samples==0]=NA
    ## matrix for mean of conditions
    mean.matrix=matrix(NA, ncol = samplen, nrow=length(area.samples[,1]),
byrow = F)
    if (pnames==TRUE){
      rownames(mean.matrix)=report$Description
    }
    else{rownames(mean.matrix)=report$Accession}
    #matrix for sd of conditions #NOT USED
    sd.matrix=matrix(NA, ncol = samplen, nrow=length(area.samples[,1]),
byrow = F)
    if (pnames==TRUE){
      rownames(sd.matrix)=report$Description
    }
    else{rownames(sd.matrix)=report$Accession}
    #counter conditions
    i.sample=seq(from=1,to=(samplen*nrep), by=nrep)
    #mean apply for each sample in nrep
    ii=0
    for (i in i.sample){
      ii=ii+1
      mean.sample=apply(area.samples[i:(i+2)],1,mean,
na.rm=TRUE)##1.2omiting NAs, NA=0 if NA mean with others
      mean.matrix[,ii]=mean.sample
    }
    #sd aplly for each sample in nrep #NOT USED
    ii=0
    for (i in i.sample){
      ii=ii+1
      sd.sample=apply(area.samples[i:(i+2)],1,sd, na.rm=TRUE)##1.2omiting
NAs, NA=0 if NA mean with others
      sd.matrix[,ii]=sd.sample
    }
    #comb 2x2 of conditions
    combinat=combn(samplen,2)
    ncomb=(factorial(samplen)/factorial(samplen-2))/2
    # grid for combinations
    gridn=PrimeSumRec(ncomb)

    #graph1
    x11() # graph of unsorted log2mean
```

```
    par(mfrow=c(gridn[2],gridn[1]))
    for (i in 1:ncomb){
      ratio=log2(mean.matrix[,combinat[2,i]]/mean.matrix[,combinat[1,i]])
      plotname=paste(combinat[2,i],":", combinat[1,i] )
      ratio=matrix(ratio)
      median.ratio=apply(ratio,2,median,na.rm=T)
      ratio=ratio-median.ratio # median norm
      ratio=sort(ratio, decreasing=FALSE)
      plot(ratio, main=plotname, col=ifelse(ratio>0, "#66C2A5",
ifelse(ratio<0, "#FC8D62", "black")), ylim=c(-10,6))
      abline(h=0)
    }
    par(mfrow=c(1,1))
   # Protein names or Accession numbers?
     ratio.matrix=matrix(NA, ncol = ncomb, nrow=length(area.samples[,1]),
byrow = F)
    if (pnames==TRUE){
      rownames(ratio.matrix)=report$Description
    }
    else{rownames(ratio.matrix)=report$Accession}
    #graph2
    x11() # graph of sorted log2mean
    par(mfrow=c(gridn[2],gridn[1]))
    for (i in 1:ncomb){
      ratio=log2(mean.matrix[,combinat[2,i]]/mean.matrix[,combinat[1,i]])
      plotname=paste(combinat[2,i],":", combinat[1,i] )
      ratio=matrix(ratio)
      median.ratio=apply(ratio,2,median,na.rm=T)
      ratio=ratio-median.ratio # median norm
      ratio.matrix[,i]=ratio
      plot(ratio, main=plotname, col=ifelse(ratio>0, "#66C2A5",
ifelse(ratio<0, "#FC8D62", "black")),ylim=c(-8,8))
    }
    par(mfrow=c(1,1))
    #graph3
    x11()#heatmap
    #housekeeping normalization & pnames check
    if (pnames==FALSE & hasArg(housek)){
      housek=paste(housek)
      hknorm=mean.matrix[housek,]
      mean.matrix=sweep(mean.matrix,2,hknorm, "/")
    }
    if (pnames==TRUE & hasArg(housek)){
      stop("Please use pnames=FALSE to consider housekeeping norm")
    }
    else{}
    # normalization and quantile shown in heatmap
    mean.matrix.heat=mean.matrix
    mean.matrix.heat[is.nan(mean.matrix.heat)]=0
    percqt=heat.qt*quantile(rowSums(mean.matrix.heat))[5] #perc of
```

```
normalized mean of last quantile
    mean.matrix.heat=mean.matrix.heat[rowSums(mean.matrix.heat)>percqt,]
#perc of normalized mean of last quantile to not appear in heatmap
    heatmap.2(mean.matrix.heat,
              notecol="black",      # change font color of cell labels to
black
              density.info="none",  # turns off density plot inside color
legend
              trace="none",         # turns off trace lines inside the heat
map
              margins =c(12,15),     # widens margins around plot
              col=brewer.pal(9,"YlOrRd"),       # use on color palette
defined earlier
              Colv="NA")
  }
  if (pnames==FALSE & hasArg(housek)){
    return(mean.matrix) #return a matrix with mean of each protein of each
sample = NaNs means Na in all reps.
  }
  else{
    return(ratio.matrix) # return ratio if housekeeping normalization was
used
  }
}
```