multQuantR

```
multQuantR <- function(map, amost, raio.esc= NULL, nome.esc= NULL,
classe.uso= unique(map@data$Classe)) #1 Cria a função "multQuantR" que
contém os argumentos "map", "amost", "raio.esc", "nome.esc" e "classe.uso".
{
  ##VERIFICANDO OS PARÂMETROS
  if(class(map)[1] != "SpatialPolygonsDataFrame") #2 O objeto "map" não é um
SpatialPolygonsDataFrame?
  {
    stop("O objeto map precisa ser um SpatialPolygonsDataFrame") #3 Se for
verdade, interrompe a função e exibe a mensagem ao usuário.
  }
  if(!"Classe" %in% colnames(map@data)) #4 O data.frame do objeto "map" não
contém uma coluna com o nome "Classe"?
  {
    stop("O data.frame do objeto map precisa ter uma coluna Classe") #5 Se
for verdade, interrompe a função e exibe a mensagem ao usuário.
  }
  if(class(map@data$Classe) != "character") #6 A coluna "Classe" do
data.frame do objeto "map" não pertence à classe "character"?
  {
    stop("O objeto map precisa ter uma coluna Classe da classe character")
#7 Se for verdade, interrompe a função e exibe a mensagem ao usuário.
  }
  if(class(amost) != "SpatialPointsDataFrame") #8 O objeto "amost" não é um
SpatialPointsDataFrame?
  {
    stop("O objeto amost precisa ser um SpatialPointsDataFrame") #9 Se for
verdade, interrompe a função e exibe a mensagem ao usuário.
  }
  if(!"Amostra" %in% colnames(amost@data)) #10 O data.frame do objeto
"amost" não contém uma coluna com o nome "Amostra"?
  {
    stop("O data.frame do objeto amost precisa ter uma coluna Amostra") #11
Se for verdade, interrompe a função e exibe a mensagem ao usuário.
  }
  if(class(amost@data$Amostra) != "character") #12 A coluna "Amostra" do
data.frame do objeto "amost" não pertence à classe "character"?
  {
    stop("O objeto amost precisa ter uma coluna Amostra da classe
character") #13 Se for verdade, interrompe a função e exibe a mensagem ao
usuário.
  }
  if(map@proj4string@projargs != amost@proj4string@projargs) #14 Os sistemas
de coordenadas dos objetos "map" e "amost" não são os mesmos?
  {
    stop("Os sistemas de coordenadas dos objetos map e amost devem ser
idênticos") #15 Se for verdade, interrompe a função e exibe a mensagem ao
usuário.
```

```
  }
  if(class(raio.esc) != "numeric") #16 O objeto "raio.esc" não pertence à
classe "numeric"?
  {
    stop("O objeto raio.esc precisa ser da classe numeric") #17 Se for
verdade, interrompe a função e exibe a mensagem ao usuário.
  }
  if(class(nome.esc) != "character") #18 O objeto "nome.esc" não pertence à
classe "character"?
  {
    stop("O objeto nome.esc precisa ser da classe character") #19 Se for
verdade, interrompe a função e exibe a mensagem ao usuário.
  }
  if(class(classe.uso) != "character") #20 O objeto "classe.uso" não
pertence à classe "character"?
  {
    stop("O objeto classe.uso precisa ser da classe character") #21 Se for
verdade, interrompe a função e exibe a mensagem ao usuário.
  }
  ##CARREGANDO OS PACOTES NECESSÁRIO
  if(sum(installed.packages()[,1] == "rgdal") == 0) #22 O pacote "rgdal" não
está instalado?
  {
    stop("Pacote rgdal não encontrado, favor instalar") #23 Se for verdade,
interrompe a função e exibe a mensagem ao usuário.
  }
  else {require(rgdal)} #24 Do contrário, carrega o pacote "rgdal".
  if(sum(installed.packages()[,1] == "rgeos") == 0) #25 O pacote "rgeos" não
está instalado?
  {
    stop("Pacote rgeos não encontrado, favor instalar") #26 Se for verdade,
interrompe a função e exibe a mensagem ao usuário.
  }
  else {require(rgeos)} #27 Do contrário, carrega o pacote "rgeos".
  if(sum(installed.packages()[,1] == "raster") == 0) #28 O pacote "raster"
não está instalado?
  {
    stop("Pacote raster não encontrado, favor instalar") #29 Se for verdade,
interrompe a função e exibe a mensagem ao usuário.
  }
  else {require(raster)} #30 Do contrário, carrega o pacote "raster".
  if(sum(installed.packages()[,1] == "reshape") == 0) #31 O pacote "reshape"
não está instalado?
  {
    stop("Pacote reshape não encontrado, favor instalar") #32 Se for
verdade, interrompe a função e exibe a mensagem ao usuário.
  }
  else {require(reshape)} #33 Do contrário, carrega o pacote "reshape".
  ##INICIANDO A CRIAÇÃO DA FUNÇÃO
  map <- map[map@data$Classe %in% classe.uso,] #34 Selecionando as classes
```

```
de cobertura definidas pelo usuário.
  lista <- list() #35 Cria uma lista vazia.
  ##QUANTIFICAÇÃO DA ÁREA OCUPADA PELAS CLASSE DE USO DO SOLO EM TORNO DAS
UNIDADES AMOSTRAIS
  for(i in 1:length(raio.esc)) #36 Inicia um "looping" por meio da função
"for" com contador i que vai de 1 até o comprimento do vetor "raio.esc".
  {
    buf <- buffer(amost, width= raio.esc[i], dissolve=F) #37  Cria "buffers"
circulares em torno das unidades amostrais (objeto amost) com raio igual à
posição i do vetor "raio.esc".
    inters <- intersect(map, buf) #38 Cria um novo SpatialPolygonsDataFrame
contendo a intersecção do mapa de cobertura com os "buffers" circulares.
    inters$Area_ha <- round((area(inters) / 10000), 2) #39 Calcula a área em
hectares ocupada por cada polígono contido no objeto "inters" e armazena na
coluna "Area_ha".
    sum.area <- aggregate(Area_ha ~ Classe + Amostra, inters, FUN= sum) #40
Agrega por soma a área dos polígonos (coluna Area_ha) do objeto inters em
função das classes de cobertura do solo (coluna Classe) e das unidades
amostrais (coluna Amostra).
    df.melt <- melt(sum.area, id.var=c("Amostra", "Classe"),
measure.var="Area_ha") #41 Transforma a coluna "Area_ha" do objeto
"sum.area" nas colunas "variable" (nome original da coluna; "Area_ha") e
"value" (valor de área de cada polígono).
    df.cast <- cast(df.melt, Amostra ~ Classe, fun.aggregate = sum) #42
Remodela o objeto "df.melt" de maneira que cada unidade amostral (linhas)
receba a soma da área de suas respectivas classes de cobertura (colunas).
    colnames(df.cast)[-1] <- paste(colnames(df.cast)[-1], nome.esc[i], sep =
"_") #43 Adiciona em todas as colunas, exceto na primeira ("Amostra"), o
nome da escala espacial presente na posição i do vetor "nome.esc".
    lista[[i]] <- df.cast #44 Salva cada data.frame df.cast na posição i do
objeto "lista".
  } #45 Termina o "looping" criado pela função "for" após executar os i
ciclos.
  lista[[1+length(raio.esc)]] <- amost@data["Amostra"] #46 Adiciona, no
final do objeto "lista", um data.frame contendo o nome de todas as unidades
amostrais (coluna "Amostra").
  df.merg <- merge_recurse(lista) #47 Mescla recursivamente todos os
data.frames do objeto "lista" com base no nome das unidades amostrais
(coluna "Amostra"; única coluna comum a todos os data.frames).
  df.merg.lin.ord <- df.merg[order(df.merg$Amostra),] #48 Reordena
alfabeticamente as linhas do objeto "df.merg" com base no nome das unidades
amostrais (coluna "Amostra").
  df.merg.col.sort <- df.merg.lin.ord[, c(colnames(df.merg.lin.ord)[1],
sort(colnames(df.merg.lin.ord)[-1]))] #49 Reordena alfabeticamente as
colunas do objeto "df.merg.lin.ord", mantendo a coluna "Amostra" na primeira
posição.
  df.merg.col.sort[is.na(df.merg.col.sort)] <- 0 #50 Substitui todos os NAs
do objeto "df.merg.col.sort" por zero (0). Isso é necessário porque
dependendo do tamanho da escala e do local do mapa de cobertura, pode ser
que não
                                                    #haja polígonos de alguma
```

```
classe de interesse. Logo, não haverá valor de área para essas unidades
amostrais em relação à essas classes e a função merge_recurse adicionará
NAs.
  return(df.merg.col.sort) #51 Retorna o objeto "df.merg.col.sort" contendo
o cálculo de área para cada classe de cobertura e escala espacial de
interesse.
} #52 Termina a função multQuantR.
```