Ricardo Bertoncello

Doutorando em Ecologia, Instituto de Biociências, USP.

O título de minha tese é: “Restauração Ecológica De Áreas Degradadas E Processos Estruturadores De Comunidades Vegetais”, orientado pela Prof. Adriana Martini.

Meus Exercícios

exercicio aula 1 exercicio aula 4 ex5.r 7_b.r 7_2.r 8_2.r

Proposta de Trabalho Final

Plano A

A elaboração de extensas planilhas de dados está sujeitas a diversos tipos de erro, que podem estar associados à tomada das mediadas em campo ou ao planilhamento dos dados. Dessa maneira, o objetivo dessa função é realizar uma busca por dados fora de padrão e apresentar possíveis correções e recomendações de novas conferências em campo.

Argumentos:

Dados: data.frame que contenha um conjunto de indivíduos monitorados, associados à variáveis lógicas (ex: vivo ou morto) e variáveis contínuas (ex: altura, diâmetro, projeção de copa e etc).

Classes-haverá um argumento para cada tipo de classe. Nesse argumento devem aparecer as colunas associadas a cada classe (ex: lógica, numérica, caractere e fator)

Outlier- Nesse argumento o usuário escolhe entre verdadeiro ou falso. Caso verdadeiro, serão apresentados todos os outliers das colunas que contenham classes numéricas.

Miracle- Detecta quando o indivíduo é considerado morto e volta a ser considerado vivo

Saída: Lista de possíveis erros relacionados aos diferentes argumentos

Lista de indivíduos que precisam ser verificados em campo, e o que precisa ser checado.

Plano B

Dada uma quantidade inicial de malte na elaboração de cervejas, existe um trade off entre a densidade de corpo e a quantidade de álcool. Esse trade off é condicionado pela temperatura e pela duração do cozimento do mosto. O objetivo dessa função é estabelecer qual a temperatura e tempo de cozimento do mosto e prever (i) a porcentagem de álcool, dada a densidade do corpo e a

quantidade inicial de malte, (ii) quantidade de malte necessário, dada a porcentagem de álcool e a densidade do corpo, e (iii) densidade do corpo dada a quantidade inicial de malte e a porcentagem de álcool que o usuários pretende obter em sua cerveja.

Argumentos:

Litros: quantos litros se pretende produzir

Malte: quantidade inicial de malte

Álcool: quantidade de álcool desejado

Corpo: densidade desejada do corpo da cerveja

Saída:

i)Dada a quantidade de cerveja que se pretende produzir, a quantidade inicial de malte e a densidade do corpo da cerveja, a função retorna o tempo e temperatura de cozimento e a porcentagem de álcool que a cerveja ficará.

ii)Dada a quantidade de cerveja que se pretende produzir, a densidade do corpo da cerveja e a porcentagem de álcool que se pretende obter, a função retorna o tempo e temperatura de cozimento e a quantidade inicial de malte necessária.

iii) Dada a quantidade de cerveja que se pretende produzir, a quantidade inicial de malte e a quantidade desejada de álcool, a função retorna o tempo e temperatura de cozimento e a densidade do corpo que a cerveja ficará.

Caso o usuário indique mais do que dois argumentos além da quantidade que se pretende produzir, será retornado um aviso de erro.

> Ricardo, as duas propostas estão bem estruturadas. A proposta A parece bem útil e do jeito que vc pensou não fica restrito a um grupo, o que é bom. Não esqueça de definir bem os argumentos no help (principalmente como o outlier vai funcionar). Eu não entendi bem o funcionamento do miracle, mas se vc já pensou como vai funcionar, manda ver! A proposta B é divertida, porém mais simples.
>
> —- *Sheina*
>
> Concordo plenamente com os comentários da Sheina... apesar de ser fã da cerveja! Bom trabalho. — *Alexandre Adalardo de Oliveira 2014/04/25 19:13*

função plancor

```
plancor<- function(dados,num=0,int=0,fact=0, logic=0,char=0, out=0)#o
usuário entra com o número das colunas do data frame que devem ser de cada
uma das classes
{
  #####argumento num####
  if(num==0){saida.num<-cat(" não há argumento numérico selecionado - ")}#se
o usuário não definir argumento numerico, o default da função é 0
  else{ #se o argumento numerico for definido
 numerico<-dados[,num]#objeto para guardar os dados originais das colunas
que devem ser numericas
 saida.num<-vector("list", length(num))#cria lista onde serão guardadas as
informações dos erros
 names(saida.num)<-names(dados[num])#identifica as dimensões da lista

 for (i in 1:length(num))#loop para as colunas que devem ser numericas
 {
   dados[,num[i]]<-as.numeric(dados[,num[i]])#Coerção para numerico. Neste
passo, o que for erro na planilha original será transformado em NA
   posicao.n<-which(is.na( dados[,num[i]]))#identifica a posição dos erros,
mostrando o que foi transformado em NA a partir da planilha original
if(length(num)==1)#se houver apenas uma coluna que deve ser numérica, siga
por aqui
  {
  erro.n<-numerico[posicao.n]#identifica o erro na planilha original
}
if(length(num)>1)#se houver mais de uma coluna que deve ser numérica, siga
por aqui
{
  erro.n<-numerico[posicao.n,i]#identifica o erro na planilha original
}

  saida.num[[i]]<-data.frame(posicao.n,erro.n)#guarda na lista criada
anteriormente a posição e o erro de cada uma das colunas que deveriam ser
numericas
}
}



####argumento int####
if(int==0){saida.int<-cat("   não há argumento integer selecionado - ")} #se
o usuário não definir argumento inteiro
else{#se o argumento inteiro for definido
inteiro<-dados[,int]#objeto para guardar os dados originais das colunas que
devem ser integer
saida.int<-vector("list", length(int))#cria lista onde serão guardadas as
informações dos erros
names(saida.int)<-names(dados[int])#identifica as dimensões da lista

for (i in 1:length(int))#loop para as colunas que devem ser integer
{
  dados[,int[i]]<-as.integer(dados[,int[i]])#Coerção para numerico. Neste
```

```
passo, o que for erro na planilha original será transformado em NA
  posicao.int<-which(is.na( dados[,int[i]]))#identifica a posição dos erros,
mostrando o que foi transformado em NA a partir da planilha original

if(length(int)==1)#se houver apenas uma coluna que deve ser inteira, siga
por aqui
{
    erro.int<-inteiro[posicao.int]#identifica o erro na planilha original
}

if(length(int)>1)#se houver mais de uma coluna que deve ser lógica, siga por
aqui
{
  erro.int<-inteiro[posicao.int,i]#identifica o erro na planilha original
}
saida.int[[i]]<-data.frame(posicao.int,erro.int)#guarda na lista criada
anteriormente a posição e o erro de cada uma das colunas que deveriam ser
numericas

}
}

####argumento logic####
if(logic==0){saida.logic<-cat("  não há argumento lógico selecionado -
")}#se o usuário não definir colunas do data.frame que devem ser da classe
lógica, o default do arguemnto é 0
else{ #se o argumento logic for definido
logica<-dados[,logic]#guarda os dados originais das colunas logicas
saida.logic<-vector("list", length(logic))#lista para guardar os erros
encontrados
names(saida.logic)<-names(dados[logic])#dá nome às colunas da lista

for (i in 1:length(logic))#loop para as colunas que devem ser lógicas
{
  dados[,logic[i]]<-as.logical(dados[,logic[i]])#Coerção para logical. Neste
passo, o que for erro na planilha original será transformado em NA
  posicao.log<-which(is.na( dados[,logic[i]]))#identifica a posição dos
erros, mostrando o que foi transformado em NA a partir da planilha original
if (length(logic)==1)#se houver apenas uma coluna lógica, siga por aqui
{

erro.log<-logica[posicao.log]#identifica o erro na planilha original

}

if (length(logic)>1)#se houver mais de uma coluna lógica, siga por aqui
{
  erro.log<-logica[posicao.log,i]#identifica o erro na planilha original
}
saida.logic[[i]]<-data.frame(posicao.log,erro.log)#guarda na lista criada
```

```
anteriormente a posição e o erro de cada uma das colunas que deveriam ser
logical
}
}



####argumento fact####

if(fact==0){saida.fact<-cat("  não há argumento factor selecionado - ")}
#caso o usuário não defina colunas do data.frame que devem ser da classe
factor
else{ #caso as colunas que devem ser da classe factor sejam definidas
factor<-dados[,fact] #guarda os dados originais das colunas que devem ser
factor
dados[,fact][dados[,fact]==""]<-NA #preenche com NA as células vazias das
colunas selecionadas como factor
saida.fact<-vector("list", length(fact)) #cria lista para guardar os erros
encontrados
names(saida.fact)<-names(dados[fact]) #dá nome às colunas da lista

for (i in 1:length(fact)) #loop para as colunas que devem ser factor
{
  dados[,fact[i]]<-as.factor(dados[,fact[i]])#Coerção para fator. Neste
passo, o que for erro na planilha original será transformado em NA
  posicao.fact<-which(is.na( dados[,fact[i]]))#identifica a posição dos
erros, mostrando o que foi transformado em NA a partir da planilha original
  if (length(fact)==1)#se houver apenas uma coluna de fator, siga por aqui
{
    erro.fact<-factor[posicao.fact]#identifica o erro na planilha original
}

if (length(fact)>1)#se houver mais de uma coluna que deve ser factor, siga
por aqui
{
    erro.fact<-factor[posicao.fact,i]#identifica o erro na planilha original
}
    saida.fact[[i]]<-data.frame(posicao.fact,erro.fact)#guarda na lista
criada anteriormente a posição e o erro de cada uma das colunas que deveriam
ser logical
  }
}

####argumento character####

if(char==0){saida.char<-cat(" não há argumento character selecionado - ")}
#se não houver colunas selecionada para ser character
else{ #se o usuário definir colunas do data.frame que devem ser character
character<-dados[,char]#objeto para guardar os dados originais das colunas
que devem ser characters
saida.char<-vector("list", length(char))#cria lista onde serão guardadas as
informações dos erros
```

```
names(saida.char)<-names(dados[char])#identifica as dimensões da lista

for (i in 1:length(char))#loop para as colunas que devem ser numericas
{
  dados[,char[i]]<-as.character(dados[,char[i]])#Coerção para numerico.
Neste passo, o que for erro na planilha original será transformado em NA
  posicao.char<-which(is.na( dados[,char[i]]))#identifica a posição dos
erros, mostrando o que foi transformado em NA a partir da planilha original
if (length(char)==1)#se houver apenas uma coluna character, siga por aqui
{
    erro.char<-character[posicao.char]#identifica o erro na planilha
original
}

if(length(char)>1)#se houver mais de uma coluna que deve ser character, siga
por aqui
{
    erro.char<-character[posicao.char,i]#identifica o erro na planilha
original
}
    saida.char[[i]]<-data.frame(posicao.char,erro.char)#guarda na lista
criada anteriormente a posição e o erro de cada uma das colunas que deveriam
ser numericas
  }
}

 ####argumento outlier####

if(out>0)#se o usuário definir colunas para as quais quer que sejam
identificados outliers, que podem ser erros na planilhas. Caso não seja
definidas colunas, o default deste argumento é 0
{
  numerico<-dados[,out]#objeto para guardar os dados originais das colunas
que devem ser numericas
outlier<-vector("list", length(out))#cria lista onde serão guardadas as
informações dos outliers
names(outlier)<-names(dados[out])#identifica as dimensões da lista
for (i in 1:length(out))#loop para as colunas que devem ser numericas
{
  dados[,out[i]]<-as.numeric(dados[,out[i]])#Coerção para numerico.
  outlier[i]<-boxplot.stats(dados[,out[i]],do.out=TRUE)[4]## indica os
outliers
 }
}
if(out==0){outlier<- cat(  "não é possivel usar argumento outlier sem
selecionar colunas numéricas")}
lista<-list(saida.num, saida.int, saida.fact, saida.logic, saida.char,
outlier)### junta tudo que a funçao vai retornar
names(lista)<-c("numeric", "integer", "factor", "logic", "character",
"outliers")#define nomes das colunas da lista que a função vai retornar
```

```
return(lista)#retorne para o usuário este objeto
}
```

Help função plancor

```
Plancor           Package: none      R documentation

Function to find error in data.

Description

Plancor looks for wrong data entrance. The user determines the class of each
column, and function indicates mistakes at each column and their position.
Besides, it shows outliers, that are possible errors in the data.frame.

Usage:

plancor(dados,num=0,int=0,logic=0,fact=0,char=0, out=0)

Arguments:

dados: data.frame containing a dataset with logical variables (eg: dead or
alive) and continuous variables (eg: height, diameter, tree
projection and etc)
num: this argument indicates the position of the columns that must be
numeric
int: this argument indicates the position of the columns that must be
integer
logic: this argument indicates the position of the columns that must be
logical
fact: this argument indicates the position of the columns that must be
factor
char: this argument indicates the position of the columns that must be
character
out: this argument indicates the number of the columns where outliers should
be located.

Details

User must define the class of each column
Argument out is not available for non-numerical classes. This argument
indicates the values of any data points which lie beyond the extremes of the
whiskers of a boxplot with the provided data.

Values

List of data.frames with mistakes at each data column and their position
Values of any numerical data points which lie beyond the extremes of the
whiskers
```

```
Author

Ricardo Bertoncello

Examples:

require(datasets)
data(iris)
head(iris)
str(iris)
mudar1<-sample(1:150, 10)
mudar2<-sample(1:150,10)
mudar3<-sample(1:150,4)
iris[mudar1,1]<-c("NA", "", "2,2", 10000, -300)
iris[mudar2,2]<-NA
iris[mudar3,5]<-c(NA, "")
erros.em.iris<-plancor(dados=iris,num=c(1:4),fact=5,out=1)
erros.em.iris

ind<-c(1:100)
bloco<-rep(c(1:10),each=10)
tratamento<-rep(c("a","b","c","d","e"), each=20)
cod.sp<-rep(c("E.umb", "P. cat", "S.par", "A. gla", "E.edu", "T. mic",
"A.col","E.rob", "T. mut", "C. pac."), each=10)
sobr<- rep(c(TRUE, TRUE, FALSE, TRUE, TRUE), each= 20)
h<-c(rnorm(98, mean=2), 100000, 500)
exemplo<-data.frame(ind, bloco, tratamento, cod.sp, sobr, h)
str(exemplo)
erros.em.exemplo<-plancor(dados=exemplo, num=6, fact=c(3), char=4,
logic=5,int=c(1,2), out=6)
erros.em.exemplo
```

script da função

plancor.r