Diego Pereira Nogueira da Silva

Mestrando pelo Departamento de Fisiologia (IB-USP)

O título de minha dissertação é: “Análise do comportamento de rã touro (*Lithobates catesbeianus*): personalidade, síndromes comportamentais e efeito da exposição crônica ao estresse térmico sobre o comportamento”, sob orientação do Prof. Dr. Fernando Ribeiro Gomes.

Meus Exercícios

No link: Exercícios

Proposta de Trabalho Final

Proposta A: Uma função que calcule os valores individuais (IVs) de seus Pokémon

---

Contextualização:

Pokémon (também conhecidos como Pocket Monsters, “monstros de bolso”, em Português), é uma antiga franquia de domínio da Nintendo. Criada por Satoshi Tajiri em 1996, inicialmente tratava-se apenas de um jogo de Game Boy, o qual possuía duas versões, conhecidas como Pokémon Red e Pokémon Blue.

A franquia cresceu muito nos últimos 23 anos, com a criação, além dos jogos, também de um mangá e uma série animada. Ademais, atualmente somam-se 809 espécies Pokémon descritas, as quais podem ser acessadas em um acervo virtual (devidamente categorizadas pelos seus atributos físicos, distribuição populacional, dentre outras características de interesse), partindo-se de uma descrição de apenas 151 espécies, um grande avanço para os taxônomos do grupo.

À esquerda, Pika electricus (Tajiri, 1996), também conhecido como Pikachu). À direita: Outra espécie de roedor também popularmente conhecida como Pika (Ochotona sp.), não confundir.

Porém, em todas as diferentes mídias, a ideia geral se mantém a mesma. Na animação e no mangá,

acompanhamos a vida de diversos treinadores Pokémon. Esse treinadores são pessoas que saem a campo para capturar diversos Pokémon selvagens, treiná-los e usá-los em batalhas. Pode parecer algo horrível e cruel para um telespectador desavisado, porém, na maioria dos casos, os Pokémon são muito bem cuidados e escolhem ir com o treinador por vontade própria.

Nos jogos, a grande diferença se dá no fato de que o até então telespectador tem a capacidade de se tornar um treinador de Pokémon, graças a uma interface interativa. Esses treinadores se dividem em diversos tipos: desde os mais casuais aos mais competitivos, os quais buscam os melhores Pokémon possíveis para completarem seus times, de forma a terem uma pequena, porém signficativa, vantagem em suas batalhas. Dentre as mecânicas que os jogadores competitivos utilizam, é possível destacar a seleção de Pokémon com base nos Valores Individuais (IV) de cada um dos seus atributos (Vida, Ataque, Defesa, Ataque Especial, Defesa Especial e Velocidade), segundo as fórmulas:

Para tal, esses treinadores utilizam-se de diversos programas para descobrirem os IVs de seus Pokémon, de forma a saber se eles estão aptos à entrarem em batalhas competitivas, a função proposta visa auxiliar neste processo.

Planejamento da Função:

Entrada: `iv.calc(pokemon, stats, EV = c(0, 0, 0, 0, 0, 0), lvl, nature = "Hardy", database = base.stats)`

- `database` = base stats para cada Pokémon que entrarão na fórmula posta acima (Base). *[class: dataframe, [,1] = character, [,(2:7)] = integer]*

Para 1 Pokémon

- `pokemon` = espécie de Pokémon de interesse para cálculo dos IVs, conforme uma lista com 809 espécies, além de suas variações (e.g.: Mega Charizard, Alola Sandshrew). *[class: character]*
- `stats` = vetor numérico contendo 6 valores (um para cada atributo do Pokémon, devem vir na ordem correta (HP, ATK, DEF, Sp.A, Sp.D, SPD) para evitar erros futuros. *[class: integer, 1 ≤ stats ≤ 255]*
- EV = vetor numérico contendo 6 valores referente ao valor de esforço para cada atributo do Pokémon de interesse, devem vir na ordem correta (HP, ATK, DEF, Sp.A, Sp.D, SPD) para evitar erros futuros. *[class: integer, 0 ≤ EV ≤ 255, sum(EV) ≤ 510]*
- `lvl` = nível do Pokémon de interesse. *[class: integer, 1 ≤ lvl ≤ 100]*
- `nature` = natureza do pokémon, sendo que elas podem ser positivas, neutras ou negativas para cada atributo, adicionando um multiplicador de 1.1, 1 ou 0.9, respectivamente. *[class: factor]*

Para diversos Pokémon

- `pokemon` = coluna de um dataframe com n Pokémon. *[class: character]*
- `stats` = seis colunas de um dataframe com n Pokémon. *[class: integer, 1 ≤ stats ≤ 255]*
- EV = seis colunas de um dataframe com n Pokémon. *[class: integer, 0 ≤ EV ≤ 255, sum(EV) ≤ 510]*

- `lvl` = coluna de um dataframe com n Pokémon. *[class: integer, 1 ≤ lvl ≤ 100]*
- `nature` = coluna de um dataframe com n Pokémon. *[class: factor]*

Verificando os parâmetros:

- `pokemon` contém apenas nomes de Pokémon que constam no database? Se não:
  - *1 Pokémon*: stop("Esse Pokémon parece não existir, verifique se digitou corretamente")
  - *Dataframe*: warning("Algumas linhas não rodaram conforme o esperado, verifique se o nome do Pokémon foi digitado corretamente")
- `stats`, `lvl` e EV contém apenas números inteiros dentro de suas respectivas delimitações (acima)? Se não:
  - *1 Pokémon*: stop("A variável X deve ser um número inteiro ente Y e Z")
  - *Dataframe*: warning("Algumas linhas não rodaram conforme o esperado. A variável X deve ser um número inteiro entre Y e Z. Verifique se digitou corretamente")
- `stats` fornecidos condizem com os possíveis atributos para aquele Pokémon, segundo as fórmulas acima? (IMPORTANTE: 0 ≤ IV ≤ 31). Se não:
  - *1 Pokémon*: stop("Os valores fornecidos são impossíveis! Verifique se os atributos, EVs, nível e natureza estão corretos")
  - *Dataframe*: warning("Algumas linhas contêm valores impossíveis! Verifique se os atributos, EVs, níveis e naturezas estão corretos")
- `stats` e EV possuem os tamanhos apropriados? Se não:
  - *1 Pokémon*: stop("O vetor stats deve conter 6 valores inteiros")
  - *Dataframe*: stop("O vetor stats deve conter 6 colunas de um dataframe")
- `nature` condiz com as 25 possíveis naturezas para os Pokémon? Se não:
  - *1 Pokémon*: stop("A natureza fornecida não existe, verifique se digitou corretamente")
  - *Dataframe*: warning("Algumas linhas não rodaram conforme o esperado. Verifique se as naturezas fornecidas estão corretas")
- `database` é um data frame contendo 7 colunas, sendo a primeira coluna caracteres e as demais valores inteiros? Se não: stop("Parece haver algum problema com seu database")

Pseudo-código:

1. Verifica se o objeto `base.stats` existe, se sim, usa ele como database, se não, cria o database a partir de um arquivo que estará na internet. #Isso permite com que o usuário faça alterações no database, caso queira;
2. Verifica se os valores fornecidos são um vetor ou dataframe;
3. Cria o objeto `valores.iv` aplicando a fórmula fornecida e arredondando para o valor inteiro mais próximo;
4. Cria o objeto `valores.iv.spread` aplicando a fórmula, simulando quais valores de IV (de 0 a 31), gerariam o mesmo valor para o atributo observado. #Isso é importante pois quanto menor o nível do Pokémon, menor a variação nos seus atributos. Ou seja, quanto menor o nível do Pokémon, mais impreciso é o cálculo dos IVs;
   1. Para tal, cria-se um ciclo `for` com contador `IV` de 0 a 31;
   2. Aplica-se a fórmula dentro do ciclo for, tomando como base para o atributo o valor que foi fornecido pelo usuário;

3. O range dos valores de IV que atingirem o mesmo valor do atributo são salvos no objeto `valores.iv.spread` e fecha-se o ciclo.

Saída:

Será retornado: `valores.iv`, `valores.iv.spread`

Caso o objeto de entrada seja um vetor, os valores de saída serão vetores. Tal qual, caso a entrada seja um dataframe, os valores de saída também serão um dataframe.

Proposta B: Criando um jogo de Escape Room

Contextualização:

Escape Room é um gênero de jogos onde o objetivo do jogador é resolver uma série de puzzles e/ou tomar decisões lógicas com o objetivo de escapar de uma sala/situação qualquer. Embora a ideia original seja mais antiga, o gênero se popularizou no meio flash em 2004 com o point-and-click entitulado Crimson Room, de Toshimitsu Takagi.

Crimson Room de Toshimitsu Takagi, se você foi criança nos anos 90/2000, você provavelmente o conhece.

Tal qual programar, jogos Escape Room podem ser bastante frustrantes inicialmente. Porém, a satisfação de chegar ao fim muitas vezes é tão boa quanto a sensação de um código que funciona. Dessa forma, minha proposta B visa criar um jogo Escape Room que seja puramente text-based, em linguagem R.

Planejamento da função:

Entrada: `escape(idioma = pt, dificuldade = normal)`

- `idioma` = permite o usuário alterar o idioma do jogo *[class: character]*
- `nivel` = permite o usuário selecionar a dificuldade do jogo *[class: character]*

#Sendo um jogo, os parâmetros apenas definem o que seriam as configurações. Uma vez iniciada a função, ela aguardaria demais inputs do usuário.

Verificando os parâmetros:

`idioma` condiz com um dos idiomas disponíveis? Se não: warning("O idioma selecionado não condiz com as opções disponíveis no momento. Iniciando o jogo em português.")

`dificuldade` condiz com uma das dificuldades disponíveis? Se não: warning("A dificuldade selecionada não condiz com as opções disponíveis no momento. Iniciando o jogo na dificuldade normal")

Pseudo-código:

1. Instala o pacote dplyr e dependências caso não esteja instalado
2. `library(dplyr)`
3. Com a função `case_when` (pacote dplyr), cria a história do jogo e as diversas opções que o jogador pode tomar, as quais aparecerão no console com a função `readline`, permitindo que o jogador interaja e tome decisões.

Saída:

Caso o jogador perca, retorna mensagem de GAME OVER. Caso vença, uma mensagem de vitória.

Trabalho Final

Proposta escolhida: A - Calculadora de Valores Individuais de Pokémon

---

Database (arquivo necessário para rodar a função)

base.stats

A função automaticamente irá baixar o database a partir deste link.

Arquivo da função

iv.calc.r

Arquivo do help

help.iv.calc.txt

Código da função

```
#Iniciando a função e definindo os parâmetros, tais quais seus valores
padrão.
iv.calc = function(pokemon, stats, EV, lvl, nature = "Hardy", interesse =
NULL, database = base.stats)
{
  #Lança a seguinte mensagem na tela, falando sobre a função e sobre como
usá-la corretamente.
  cat("ATENÇÃO! Esta função utiliza os base stats dos Pokémon com base na
sétima geração em diante.\nComo padrão, esse database é criado a partir de
um link na internet com o nome base.stats\nCaso queira utilizar um database
próprio, certifique-se de que:\n1) A primeira coluna do database contém os
nomes dos Pokémon;\n2) As demais colunas contêm seus atributos na ordem: HP,
ATK, DEF, Sp.A, Sp.D e SPD.\nDo contrário, a função irá gerar resultados
errados.\nAdemais, certifique-se também de oferecer os 6 atributos e EVs dos
Pokémons nesta mesma ordem.\n\n")

#########################################################
########## Checando e ajustando o database ##########
#########################################################
  #Checa se o arquivo padrão "base.stats" existe.
  if(exists("base.stats")==FALSE)
  {
    #Caso o arquivo não exista, cria ele a partir de um link da internet, o
argumento "as.is=T" faz com que a função não atribua fatores a caracteres.
    base.stats =
read.csv(url("http://ecologia.ib.usp.br/bie5782/lib/exe/fetch.php?media=bie5
782:01_curso_atual:alunos:trabalho_final:diego.pereira.silva:base.stats.csv"
), as.is=T)
  }
  #Caso a pessoa queira fornecer um database com outro nome, checa se este
database é um data frame.
  if(is.data.frame(database)==FALSE)
  {
    #Caso não seja, para a função e fornece uma mensagem explicativa.
    stop("Database deve ser um data frame.")
  }
#####################################################
########## Crindo a matriz de naturezas ##########
#####################################################
  #Cria a matriz de naturezas, com os respectivos valores que vão entrar nas
```

```
contas mais para frente como multiplicadores.
  nat.mat = matrix(c(rep(1,
5),1.1,0.9,1,1,1,1.1,1,0.9,1,1,1.1,1,1,0.9,1,1.1,1,1,1,0.9,0.9,1.1,1,1,1,rep
(1,
5),1,1.1,0.9,1,1,1,1.1,1,0.9,1,1,1.1,1,1,0.9,0.9,1,1.1,1,1,1,0.9,1.1,1,1,rep
(1,
5),1,1,1.1,0.9,1,1,1,1.1,1,0.9,0.9,1,1,1.1,1,1,0.9,1,1.1,1,1,1,0.9,1.1,1,rep
(1,
5),1,1,1,1.1,0.9,0.9,1,1,1,1.1,1,0.9,1,1,1.1,1,1,0.9,1,1.1,1,1,1,0.9,1.1,rep
(1, 5)),
                    #Define que a matriz tem 5 colunas e que deve ser
preenchida por linhas.
                    ncol=5, byrow=T,
                    #Adiciona os nomes das linhas (naturezas) e colunas
(atributos) da matriz.
dimnames=list(c("Hardy","Lonely","Adamant","Naughty","Brave","Bold","Docile"
,"Impish","Lax","Relaxed","Modest","Mild","Bashful","Rash","Quiet","Calm","G
entle","Careful","Quirky","Sassy","Timid","Hasty","Jolly","Naive","Serious")
, c("ATK","DEF","Sp.A","Sp.D","SPD")))

################################################
########## Definindo critérios gerais ##########
################################################
  #Checa os valores do parâmetro "stats".
  if(any(stats < 1) | any(stats > 714))
  {
    #Caso os valores ultrapassem os limites perimitidos, para a função e
fornece uma mensagem explicativa.
    stop("Os valores dos atributos devem ser números inteiros entre 1 e 714.
Pelo menos um dos atributos está fora do limite.")
  }
  #Checa os valores do parâmetro "EV".
  if(any(EV < 0 | EV > 252))
  {
    #Caso os valores ultrapassem os limites perimitidos, para a função e
fornece uma mensagem explicativa.
    stop("Os valores de EVs devem ser números inteiros entre 0 e 252. Pelo
menos um dos EVs está fora do limite.")
  }

############################################
########## Para apenas um pokémon ##########
############################################
  #Checa se o usuário forneceu apenas um pokémon.
  if(length(pokemon)==1)
  {
    #Em caso positivo, checa se "stats" e "EV" são vetores.
    if(is.vector(stats)==FALSE | is.vector(EV)==FALSE)
    {
      #Caso não sejam, para a função e fornece uma mensagem explicativa.
      stop("Os atributos e EVs devem ser um vetor numérico.")
```

```
    }
    #Checa se o pokémon fornecido existe no database.
    if(pokemon%in%database[,1]==FALSE)
    {
      #Caso não, para a função e fornece uma mensagem explicativa.
      stop("Esse Pokémon não consta no database.\nVerifique se digitou
corretamente.")
    }
    #Checa se o parâmetro "lvl" está dentro dos limites permitidos.
    if(lvl < 1 | lvl > 100)
    {
      #Caso o valor ultrapasse os limites perimitidos, para a função e
fornece uma mensagem explicativa.
      stop("O nível de um Pokémon deve ser um número inteiro entre 1 e
100.")
    }
    #Checa se a soma dos EVs está dentro do limite permitido.
    if(sum(EV) > 510)
    {
      #Caso os valores ultrapassem os limites perimitidos, para a função e
fornece uma mensagem explicativa.
      stop("A soma dos EVs de um Pokémon não deve ultrapassar 510.")
    }
    #Checa se a natureza fornecida corresponde com as naturezas existentes.
    if(nature%in%rownames(nat.mat)==FALSE)
    {
      #Caso não, para a função e fornece uma mensagem explicativa.
      stop("A natureza fornecida não corresponde com uma das 25 naturezas
possíveis.\nVerifique se digitou corretamente.")
    }
    #Cria os objetos HPs, ATKs, DEFs, Sp.As, Sp.Ds e SPDs, contendo apenas
32 NAs (pois os IVs podem ter 32 valores possíveis).
    HPs = ATKs = DEFs = Sp.As = Sp.Ds = SPDs = rep(NA, 32)
    #Inicia um for para preencher os objetos acima com os valores dos stats
que correspondem ao IV do pokémon, para aquele nível, EVs e natureza.
    for(i in 1:32)
    {
      #Preenche o objeto HPs com a respectiva fórmula para cálculo do HP do
pokémon fornecido, onde "(i-1)" corresponde ao IV, que vai de 0 a 31.
Arredondado para valores inteiros.
      HPs[i] = round(((2*database[database[,1]==pokemon,2] + (i-1) + EV[1]/4
+ 100) * lvl) / 100 + 10)
      ATKs[i] = round((((2*database[database[,1]==pokemon,3] + (i-1) +
EV[2]/4) * lvl) / 100 + 5) * nat.mat[nature,1])     #Idem acima, só que para
o atributo ATK.
      DEFs[i] = round((((2*database[database[,1]==pokemon,4] + (i-1) +
EV[3]/4) * lvl) / 100 + 5) * nat.mat[nature,2])     #Idem acima, só que para
o atributo DEF.
      Sp.As[i] = round((((2*database[database[,1]==pokemon,5] + (i-1) +
EV[4]/4) * lvl) / 100 + 5) * nat.mat[nature,3])    #Idem acima, só que para
```

```
o atributo Sp.A.
      Sp.Ds[i] = round((((2*database[database[,1]==pokemon,6] + (i-1) +
EV[5]/4) * lvl) / 100 + 5) * nat.mat[nature,4])    #Idem acima, só que para
o atributo Sp.D.
      SPDs[i] = round((((2*database[database[,1]==pokemon,7] + (i-1) +
EV[6]/4) * lvl) / 100 + 5) * nat.mat[nature,5])     #Idem acima, só que para
o atributo SPD.
    }
    #Cria o spread do atributo HP daquele pokémon fornecido, selecionando
quais valores do objeto HPs criado acima correspondem ao HP fornecido pelo
usuário em "stats".
    spread.HP = which(HPs==stats[1])
    spread.ATK = which(ATKs==stats[2])        #Idem acima, só que para o
atributo ATK.
    spread.DEF = which(DEFs==stats[3])        #Idem acima, só que para o
atributo DEF.
    spread.Sp.A = which(Sp.As==stats[4])      #Idem acima, só que para o
atributo Sp.A.
    spread.Sp.D = which(Sp.Ds==stats[5])      #Idem acima, só que para o
atributo Sp.D.
    spread.SPD = which(SPDs==stats[6])        #Idem acima, só que para o
atributo SPD.
    #Calcula o IV daquele pokémon fornecido para o atributo HP, o qual
corresponde à mediana do spread, arredondado para valores inteiros.
    iv.HP = round(median(spread.HP))
    iv.ATK = round(median(spread.ATK))        #Idem acima, só que para o
atributo ATK.
    iv.DEF = round(median(spread.DEF))        #Idem acima, só que para o
atributo DEF.
    iv.Sp.A = round(median(spread.Sp.A))      #Idem acima, só que para o
atributo Sp.A.
    iv.Sp.D = round(median(spread.Sp.D))      #Idem acima, só que para o
atributo Sp.D.
    iv.SPD = round(median(spread.SPD))        #Idem acima, só que para o
atributo SPD.
    #Cria o objeto valores IV, que corresponde a um data frame com os
valores acima.
    valores.iv = c(iv.HP, iv.ATK, iv.DEF, iv.Sp.A, iv.Sp.D, iv.SPD)
    #Verifica se os valores do objeto "valores.iv" estão corretos (caso o IV
calculado seja menor que 0 ou maior que 31, a função gera um NA, isso ocorre
caso o usuário forneça parâmetros errados).
    if(NA%in%valores.iv==TRUE)
    {
      #Caso o objeto contenha NAs, prossegue a função, mas fornece uma
mensagem de aviso, indicando que algum parâmetro fornecido estava incorreto.
      warning("Os parâmetros oferecidos geraram valores impossíveis de IV (1
<= IV <= 31).\nVerifique se a natureza e os valores dos atributos, EVs e
nível estão corretos.")
    }
    #Adiciona nomes ao vetor "valores.iv".
    names(valores.iv) = c("HP", "ATK", "DEF", "Sp.A", "Sp.D", "SPD")
```

```
    #Retorna os objetos referente aos IVs dos pokémon e seus spreads, em uma
lista.
    return(list("IVs Medianos" = valores.iv, "IV Spread - HP" = spread.HP,
"IV Spread - ATK" = spread.ATK, "IV Spread - DEF" = spread.DEF, "IV Spread -
Sp.A" = spread.Sp.A, "IV Spread - Sp.D" = spread.Sp.D, "IV Spread - SPD" =
spread.SPD))
  }
########################################
########## Para vários pokémon ##########
########################################
  #Checa se o usuário forneceu mais de um pokémon.
  if(length(pokemon)>1)
  {
    #Em caso positivo, checa se "stats" e "EV" são data frames.
    if(is.data.frame(stats)==FALSE | is.data.frame(EV)==FALSE)
    {
      #Caso não, para a função e fornece uma mensagem explicativa.
      stop("Os atributos e EVs devem ser um data frame.")
    }
    #Checa se algum dos pokémon fornecido não consta no database.
    if(any(pokemon%in%database[,1]==FALSE))
    {
      #Caso algum pokémon não conste no database, para a função e fornece
uma mensagem explicativa.
      stop("Pelo menos um Pokémon não consta no database.\nVerifique se
digitou corretamente.")
    }
    #Checa se os níveis dos pokémon estão dentro do limite permitido.
    if(any(lvl < 1 | lvl > 100))
    {
      #Caso algum valor ultrapasse o limite, para a função e fornece uma
mensagem explicativa.
      stop("O nível de um Pokémon deve ser um número inteiro entre 1 e
100.\nO nível de pelo menos um dos Pokémon está fora do limite.")
    }
    #Checa se a soma dos EVs de cada pokémon (linha) está dentro do limite
permitido.
    if(any(apply(EV, 1, sum) > 510))
    {
      #Caso algum valor ultrapasse o limite, para a função e fornece uma
mensagem explicativa.
      stop("A soma dos EVs de um Pokémon não deve ultrapassar 510.")
    }
    #Checa se alguma das naturezas fornecidas não consta na matriz de
naturezas.
    if(any(nature%in%rownames(nat.mat)==FALSE))
    {
      #Em caso positivo, para a função e fornece uma mensagem explicativa.
      stop("Pelo menos uma das naturezas fornecida não corresponde com uma
das 25 naturezas possíveis.\nVerifique se digitou corretamente.")
```

```
    }
    #Cria os objetos HPs, ATKs, DEFs, Sp.As, Sp.Ds e SPDs, os quais são
matrizes com 32 colunas (pois os IVs podem ter 32 valores possíveis) e
tantas linhas quanto o usuário fornecer de pokémon.
    HPs = ATKs = DEFs = Sp.As = Sp.Ds = SPDs = matrix(rep(NA,
length(pokemon)*32), ncol=32)
    #Inicia um for para preencher os objetos acima com os valores dos stats
que correspondem ao IV do pokémon, para aquele nível, EVs e natureza.
    for(j in 1:32)
#1:32 corresponde ao número de colunas.
    {
      #Inicia um segundo for, pois trata-se de um objeto bidimensional.
      for(i in 1:length(pokemon))
#1:length(pokemon)) corresponde ao número de linhas.
      {
        #Preenche o objeto HPs com a respectiva fórmula para cálculo do HP
do pokémon fornecido, onde "(j-1)" corresponde ao IV, que vai de 0 a 31.
Arredondado para valores inteiros.
        HPs[i,j] = round(((2*database[database[,1]==pokemon[i],2] + (j-1) +
EV[i,1]/4 + 100) * lvl[i]) / 100 + 10)
        ATKs[i,j] = round((((2*database[database[,1]==pokemon[i],3] + (j-1)
+ EV[i,2]/4) * lvl[i]) / 100 + 5) * nat.mat[nature[i],1])     #Idem acima,
só que para o atributo ATK.
        DEFs[i,j] = round((((2*database[database[,1]==pokemon[i],4] + (j-1)
+ EV[i,3]/4) * lvl[i]) / 100 + 5) * nat.mat[nature[i],2])     #Idem acima,
só que para o atributo DEF.
        Sp.As[i,j] = round((((2*database[database[,1]==pokemon[i],5] + (j-1)
+ EV[i,4]/4) * lvl[i]) / 100 + 5) * nat.mat[nature[i],3])    #Idem acima, só
que para o atributo Sp.A.
        Sp.Ds[i,j] = round((((2*database[database[,1]==pokemon[i],6] + (j-1)
+ EV[i,5]/4) * lvl[i]) / 100 + 5) * nat.mat[nature[i],4])    #Idem acima, só
que para o atributo Sp.D.
        SPDs[i,j] = round((((2*database[database[,1]==pokemon[i],7] + (j-1)
+ EV[i,6]/4) * lvl[i]) / 100 + 5) * nat.mat[nature[i],5])     #Idem acima,
só que para o atributo SPD.
      }
    }
    #Cria os objetos referentes aos spreads, os quais são n
(length(pokemon)) listas vazias.
    spread.HP = spread.ATK = spread.DEF = spread.Sp.A = spread.Sp.D =
spread.SPD = rep(list(NA),length(pokemon))
    #Cria os objetos referentes aos ivs dos pokémon, os quais são vetores
contendo length(pokemon) NAs.
    iv.HP = iv.ATK = iv.DEF = iv.Sp.A = iv.Sp.D = iv.SPD = rep(NA,
length(pokemon))
    #Inicia um for para preencher os objetos acima com seus valores.
    for(i in 1:length(pokemon))
    {
      #Preenche o spread do atributo HP daqueles pokémon fornecido,
selecionando quais valores do objeto HPs criado acima correspondem ao HP
fornecido pelo usuário em "stats".
```

```
      spread.HP[[i]] = which(HPs[i,]==stats[i,1])
      spread.ATK[[i]] = which(ATKs[i,]==stats[i,2])        #Idem acima só
que para o atributo ATK.
      spread.DEF[[i]] = which(DEFs[i,]==stats[i,3])        #Idem acima só
que para o atributo DEF.
      spread.Sp.A[[i]] = which(Sp.As[i,]==stats[i,4])      #Idem acima só
que para o atributo Sp.A.
      spread.Sp.D[[i]] = which(Sp.Ds[i,]==stats[i,5])      #Idem acima só
que para o atributo Sp.D.
      spread.SPD[[i]] = which(SPDs[i,]==stats[i,6])        #Idem acima só
que para o atributo SPD.
      #Calcula o IV daqueles pokémon fornecido para o atributo HP, o qual
corresponde à mediana do spread, arredondado para valores inteiros.
      iv.HP[i] = round(median(spread.HP[[i]]))
      iv.ATK[i] = round(median(spread.ATK[[i]]))           #Idem acima só
que para o atributo ATK.
      iv.DEF[i] = round(median(spread.DEF[[i]]))           #Idem acima só
que para o atributo DEF.
      iv.Sp.A[i] = round(median(spread.Sp.A[[i]]))         #Idem acima só
que para o atributo Sp.A.
      iv.Sp.D[i] = round(median(spread.Sp.D[[i]]))         #Idem acima só
que para o atributo Sp.D.
      iv.SPD[i] = round(median(spread.SPD[[i]]))           #Idem acima só
que para o atributo SPD.
    }
    #Cria o objeto "valores.iv", um data frame de 6 colunas, uma para cada
um dos atributos calculados acima.
    valores.iv = data.frame(Pokémon = pokemon, HP = iv.HP, ATK = iv.ATK, DEF
= iv.DEF, Sp.A = iv.Sp.A, Sp.D = iv.Sp.D, SPD = iv.SPD)
    #Verifica se os valores do objeto "valores.iv" estão corretos (caso o IV
calculado seja menor que 0 ou maior que 31, a função gera um NA, isso ocorre
caso o usuário forneça parâmetros errados).
    if(is.na(any(valores.iv==TRUE)))
    {
      #Caso o objeto contenha NAs, prossegue a função, mas fornece uma
mensagem de aviso, indicando que algum parâmetro fornecido estava incorreto.
      warning("Os parâmetros oferecidos geraram valores impossíveis de IV (1
<= IV <= 31).\nVerifique se a natureza e os valores dos atributos, EVs e
nível estão corretos.")
    }
    #Cria o objeto "temp", um arquivo temporário para os passos seguintes,
que corresponde ao objeto valores.iv, menos a primeira coluna.
    temp = valores.iv[,-1]
    #Checa se o usuário forneceu um pokémon de interesse nos parâmetros.
    if(is.null(interesse))
    {
      #Em caso negativo, cria um stripchart com os seguintes parâmetros
abaixo.
      stripchart(x=temp, #Valores de temp.
                 vertical=T,
```

```
#Na vertical.
                  method="jitter", jitter=0.2,
#Utilizando o método jitter, com 0.2 de valor de jitter.
                  cex=1.1, cex.lab=1.5, cex.axis=1.1,
#Aumenta os tamanhos dos pontos, da legenda e dos valores dos eixos,
respectivamente.
                  pch=19,
#Define o tipo de ponto do gráfico.
                  ylim=c(-2,33), xlim=c(0.5,6.5),
#Define os limites de x e y.
                  ylab="Valores Individuais",
#Define a legenda do eixo y.
                  tck=0.01,
#Define o tamanho dos tiques nos eixos.
                  las=1,
#Define a orientação do eixo.
                  family="sans",
#Define a fonte dos elementos do gráfico.
                  mgp=c(2.3,0.3,0),
#Define as distâncias da legenda, dos valores dos eixos e do eixo.
col=c("firebrick1","darkorange","goldenrod1","dodgerblue","limegreen","hotpi
nk"))    #Define as cores para os diferentes atributos dos pokémon.
      #Cria uma linha tracejada verde, com width=3, na altura 26 do eixo y,
correspondente a IVs considerado "muito bons".
      abline(26,0, col="green3", lty=3, lwd=3)
      #Retorna os objetos referente aos IVs dos pokémon e seus spreads, em
uma lista.
      return(list("IVs Medianos" = valores.iv, "IV Spread - HP" = spread.HP,
"IV Spread - ATK" = spread.ATK, "IV Spread - DEF" = spread.DEF, "IV Spread -
Sp.A" = spread.Sp.A, "IV Spread - Sp.D" = spread.Sp.D, "IV Spread - SPD" =
spread.SPD))
    }
    #Caso o usuário tenha fornecido um pokémon de interesse.
    else
    {
      #Cria o mesmo gráfico acima, com os mesmos parâmetros, exceto
removendo o pokémon de interesse dos valores plotados (x=temp[-interesse,]).
      stripchart(x=temp[-interesse,], vertical=T, method="jitter",
jitter=0.2, cex=1.1, cex.lab=1.5, cex.axis=1.1, pch=19, ylim=c(-2,33),
ylab="Valores Individuais", tck=0.01,  las=1,  family="sans",
mgp=c(2.3,0.3,0),
col=c("firebrick1","darkorange","goldenrod1","deepskyblue","limegreen","hotp
ink"))
      #Cria a mesma linha acima, correspondente a IVs considerado "muiton
bons".
      abline(26,0, col="green3", lty=3, lwd=3)
      #Plota os valores referentes ao pokémon de interesse, com formato
(pch), tamanho (cex) e cor diferentes dos demais pokémon do gráfico.
      points(x=c(1:6), y=temp[interesse,], pch=17, cex=1.5)
      #Plota o intervalo de erro (spread) para o IV do pokémon de interesse
referente a HP.
```

```
      segments(x0 = 1, y0 = min(unlist(c(spread.HP[[interesse]]))), x1 = 1,
y1 = max(unlist(spread.HP[[interesse]])), lwd=2)
      segments(x0 = 2, y0 = min(unlist(spread.ATK[[interesse]])), x1 = 2, y1
= max(unlist(spread.ATK[[interesse]])), lwd=2)       #Idem acima, só que
para o atributo ATK.
      segments(x0 = 3, y0 = min(unlist(spread.DEF[[interesse]])), x1 = 3, y1
= max(unlist(spread.DEF[[interesse]])), lwd=2)       #Idem acima, só que
para o atributo DEF.
      segments(x0 = 4, y0 = min(unlist(spread.Sp.A[[interesse]])), x1 = 4,
y1 = max(unlist(spread.Sp.A[[interesse]])), lwd=2)     #Idem acima, só que
para o atributo Sp.A.
      segments(x0 = 5, y0 = min(unlist(spread.Sp.D[[interesse]])), x1 = 5,
y1 = max(unlist(spread.Sp.D[[interesse]])), lwd=2)     #Idem acima, só que
para o atributo Sp.D.
      segments(x0 = 6, y0 = min(unlist(spread.SPD[[interesse]])), x1 = 6, y1
= max(unlist(spread.SPD[[interesse]])), lwd=2)       #Idem acima, só que
para o atributo SPD.
      #Plota o nome daquele pokémon ao lado dos seus respectivos valores no
gráfico.
      text(x=c((1:6)-.05), y=temp[interesse,]+0.5,
labels=rep(pokemon[interesse], 6), pos=4, cex=0.7)
      #Cria o objeto "ivmaior", uma lista contendo os pokémon que tenham IVs
maiores do que o pokémon de interesse para cada um dos atributos.
      ivmaior = list(HP = which(temp[,1] > temp[interesse,1]), ATK =
which(temp[,2] > temp[interesse,2]), DEF = which(temp[,3] >
temp[interesse,3]), Sp.A = which(temp[,4] > temp[interesse,4]), Sp.D =
which(temp[,5] > temp[interesse,5]), SPD = which(temp[,6] >
temp[interesse,6]))
      #Cria o objeto "ivmenor", uma lista contendo os pokémon que tenham IVs
maiores do que o pokémon de interesse para cada um dos atributos.
      ivmenor = list(HP = which(temp[,1] < temp[interesse,1]), ATK =
which(temp[,2] < temp[interesse,2]), DEF = which(temp[,3] <
temp[interesse,3]), Sp.A = which(temp[,4] < temp[interesse,4]), Sp.D =
which(temp[,5] < temp[interesse,5]), SPD = which(temp[,6] <
temp[interesse,6]))
      #Cria o objeto "ivspread", uma lista contendo os pokémon cujos IV
estejam dentro do intervalo de erro (spread) do pokémon de interesse.
      ivspread = list(HP = which(temp[,1] <=
max(unlist(spread.HP[[interesse]])) & temp[,1] >=
min(unlist(spread.HP[[interesse]]))),          #Cria a lista para o spread
de HP.
                      ATK = which(temp[,2] <=
max(unlist(spread.ATK[[interesse]])) & temp[,2] >=
min(unlist(spread.ATK[[interesse]]))),       #Cria a lista para o spread de
ATK.
                      DEF = which(temp[,3] <=
max(unlist(spread.DEF[[interesse]])) & temp[,3] >=
min(unlist(spread.DEF[[interesse]]))),       #Cria a lista para o spread de
DEF.
                      Sp.A = which(temp[,4] <=
```

```
max(unlist(spread.Sp.A[[interesse]])) & temp[,4] >=
min(unlist(spread.Sp.A[[interesse]]))),    #Cria a lista para o spread de
Sp.A.
                      Sp.D = which(temp[,5] <=
max(unlist(spread.Sp.D[[interesse]])) & temp[,5] >=
min(unlist(spread.Sp.D[[interesse]]))),    #Cria a lista para o spread de
Sp.D.
                      SPD = which(temp[,6] <=
max(unlist(spread.SPD[[interesse]])) & temp[,6] >=
min(unlist(spread.SPD[[interesse]]))))      #Cria a lista para o spread de
SPD.
      #Retorna os objetos valores.iv, ivmaior, ivmenor e ivspread, em uma
lista.
      return(list("IVs Medianos" = valores.iv, "IVs Maiores" = ivmaior, "IVs
Menores" = ivmenor, "IVs em Spread" = ivspread))
    }
  }
}
```

Código do help

```
iv.calc                                                 package:unknown
R Documentation

~~Calculadora de Valores Individuais (Pokémon)~~
  Descrição:
  ~~ iv.calc é uma função criada para calcular os valores individuais (IVS)
de seus pokémon dos jogos da geração III ~~
  ~~ em diante. Como padrão, a função usa os valores base (database) da
geração VII em diante. ~~
  ~~ Não funciona para o Pokémon GO. ~~
  Uso:
  ~~ iv.calc(pokemon, stats, EV, lvl, nature = "Hardy", interesse = NULL,
database = base.stats) ~~
  Arguments:
  ~~ pokemon     O nome de pokémon que deseja calcular o IV. Deve vir
escrito exatamente como consta o nome do ~~
  ~~             pokémon no database. Pode ser um valor único ou um vetor
contendo o nome de n pokémon. ~~
  ~~ stats       Os valores dos atributos de seus pokémon, na ordem: HP,
ATK, DEF, Sp.A, Sp.D e SPD.  ~~
  ~~             Pode ser um vetor contendo apenas esses 6 valores, ou um
data.frame contendo 6 colunas e n linhas ~~
  ~~             equivalentes a cada pokémon [1 <= stats <= 714]. ~~
  ~~ EV          Os EVs ou valores de esforço de seus pokémon, na ordem: HP,
ATK, DEF, Sp.A, Sp.D e SPD. ~~
  ~~             Pode ser um vetor contendo apenas esses 6 valores, ou um
data.frame contendo 6 colunas e n linhas ~~
```

```
  ~~             equivalentes a cada pokémon [0 <= EV <= 252; sum(EV) <
510]. ~~
  ~~ lvl         O nível de seus pokémon. Pode ser um valor único ou um
vetor contendo os níveis de n pokémon.  ~~
  ~~             [1 <= lvl <= 100]. ~~
  ~~ nature      A natureza de seus pokémon. Deve vir escrita da seguinte
forma: "Hardy","Bashful","Adamant", etc.  ~~
  ~~             Também deve conter apenas os nomes das 25 possíveis
naturezas dos pokémon. Pode ser um valor único ~~
  ~~             ou um vetor contendo as naturezas de n pokémon. ~~
  ~~ interesse   Dados vários pokémon (n > 1), o usuário pode escolher um
pokémon de interesse em ~~
  ~~             particular para ser comparado com os demais pokémon em
relação a seus IVs. Deve ser um número inteiro ~~
  ~~             único de 1 a n, correspondente a posição do pokémon de
interesse no vetor pokemon. ~~
  ~~ database    Um dataframe contendo 7 colunas, sendo a primeira coluna o
nome de todos os Pokémon e as demais  ~~
  ~~             colunas seus atributos basais, utilizados no cálculo dos
IVs, na ordem HP, ATK, DEF, Sp.A, Sp.D e SPD. ~~
  ~~             Caso o usuário não forneça um database, ele é baixado
diretamente da internet. ~~
  Detalhes:
  ~~ Fórmulas usadas nessa função: ~~
  ~~ Cálculo do HP:                     HP = ((2*ValorBasal + IV + EV/4 +
100) * Nível) / 100 + 10 ~~
  ~~ Cálculo dos demais atributos:      Atributo = (((2*ValorBasal + IV +
EV/4) * Nível) / 100 + 5) * Natureza ~~
  ~~ Caso o usuário forneça valores errados para stats, lvl, EV ou nature, a
função irá rodar, porém, poderão ser ~~
  ~~ criados valores impossíveis de IV impossíveis (0 <= IV <= 31). Nesse
caso, o usuário receberá um aviso e a saída ~~
  ~~ conterá NAs e listas vazias. ~~
  ~~ Os valores de stats variam de 1 a 714 pois esses são o menor e maior
valor que um pokémon pode ter. ~~
  ~~ Shedinja sempre possui 1 de HP e uma Blissey com 31 de IV e 252 de EVs
pode atingir 714 HP. ~~
  ~~ Por esse motivo, caso os valores ultrapassem esse limite, escolhi que a
função parasse automaticamente, ~~
  ~~ evitando a criação de NAs mais para frente. ~~
  ~~ As natureza dos pokémon entram na fórmula na forma de um multiplicador
que pode assumir os valores 1.1, 1 ou 0.9 ~~
  ~~ caso elas sejam respectivamente, "positivas", "neutras" ou "negativa"
para determinado atributo. ~~
  ~~ Por ser uma natureza neutra para todos os atributos, "Hardy" foi
colocada como padrão para a função. ~~
  ~~ Além de calcular os IVs dos pokémon, a função também calcula o spread
de seus IVs. Isto é, os valores de IV ~~
  ~~ que gerariam um valor igual ao valor do atributo do pokémon quando
arredondado para zero casas decimais. ~~
```

```
  ~~ Isso ocorre pois a fórmula do cálculo dos IVs gera números reais, porém
os IVs podem apenas ser números inteiros. ~~
  ~~ Ademais, dadas as fórmulas, quanto menor o nível dos pokémon, maior é o
erro associado ao cálculo do IV. ~~
  ~~ Caso sejam fornecidos n > 1 pokémon, a função possui uma saída gráfica
(scatterplot), onde os valores dos ~~
  ~~ IVs de cada atributo são mostrados graficamente, separados pelas cores
que eles apresentam nos jogos. ~~
  ~~ Ademais, a função também adiciona uma linha horizontal referente ao
limiar de IVs que é considerado "muito bom" ~~
  ~~ (IV >= 26). Por fim, caso o usuário forneça um pokémon de interesse em
particular, este recebe destaque no gráfico ~~
  ~~ e também virá com uma barra de erro referente aos valores de spread
possível dos seu IVs. ~~

  Value:
  ~~ Retorna uma de duas listas possíveis. ~~
  ~~ Se pokemon tiver n = 1, ou n > 1 & interesse = NULL ~~

[[1]] IVs Medianos     : Um vetor para n = 1 ou um data.frame para n > 1,
contendo os IVs de cada um dos pokémon.

[[2]] IV Spread - HP   : Um vetor para n = 1 ou uma lista para n > 1,
contendo o spread para os IVs de HP de cada um dos pokémon.

[[3]] IV Spread - ATK  : Um vetor para n = 1 ou uma lista para n > 1,
contendo o spread para os IVs de ATK de cada um dos pokémon.

[[4]] IV Spread - DEF  : Um vetor para n = 1 ou uma lista para n > 1,
contendo o spread para os IVs de DEF de cada um dos pokémon.

[[5]] IV Spread - Sp.A : Um vetor para n = 1 ou uma lista para n > 1,
contendo o spread para os IVs de Sp.A de cada um dos pokémon.

[[6]] IV Spread - Sp.D : Um vetor para n = 1 ou uma lista para n > 1,
contendo o spread para os IVs de Sp.D de cada um dos pokémon.

[[7]] IV Spread - SPD  : Um vetor para n = 1 ou uma lista para n > 1,
contendo o spread para os IVs de SPD de cada um dos pokémon.

  ~~ Se pokemon tiver n > 1 & interesse != NULL ~~

[[1]] IVs Medianos     : Um vetor para n = 1 ou um data.frame para n > 1,
contendo os IVs de cada um dos pokémon.

[[2]] IVs maiores      : Uma lista com a posição dos pokémon no parâmetro
pokemon que possuem os IVs maiores que o pokémon de
                         interesse em cada um dos atributos.

[[3]] IVs menores      : Uma lista com a posição dos pokémon no parâmetro
pokemon que possuem os IVs menores que o pokémon de
```

```
                          interesse em cada um dos atributos.

[[4]] IVs em spread    : Uma lista com a posição dos pokémon no parâmetro
pokemon que possuem os IVs dentro do intervalo de
                          spread para o pokémon de interesse em cada um dos
atributos. (Contém o próprio pokémon de interesse).


  Notas:
  ~~ O objeto de saída pode ficar extremamente poluído para um n grande de
pokémon, porém, dada a natureza dos spreads, ~~
  ~~ os quais são vetores de 0 até 32 posições, não era possível salvá-los
em um formato que não lista. ~~
  ~~ Use a indexação das listas ([[]]) acima para facilitar a vizualição. ~~
  Autor:
  ~~ Diego Pereira Nogueira da Silva ~~
  Referências:
  ~~ Bulbapedia: https://bulbapedia.bulbagarden.net/wiki/Individual_values
~~
  Examples:

 ## Exemplo para 1 pokémon
  exemplo = iv.calc(pokemon="Bulbasaur",
                    stats=c(110,69,60,80,80,50),
                    EV=c(0,0,0,0,0,0),
                    lvl=50,
                    nature="Brave")
 ## Exemplo para mais de um pokémon e interesse = NULL
  exemplo2 = iv.calc(pokemon=c("Bulbasaur","Blissey"),
stats=data.frame(HP=c(110,623),ATK=c(69,51),DEF=c(60,51),Sp.A=c(80,174),Sp.D
=c(80,285),SPD=c(50,122)),
EV=data.frame(EVHP=c(0,0),EVATK=c(0,0),EVDEF=c(0,0),EVSp.A=c(0,0),EVSp.D=c(0
,0),EVSPD=c(0,0)),
                     lvl=c(50,100),
                     nature=c("Brave","Brave"))
 ## Exemplo para mais de um pokémon de interesse != NULL
  exemplo3 = iv.calc(pokemon=c("Bulbasaur","Blissey","Pikachu"),
stats=data.frame(HP=c(110,623,166),ATK=c(69,51,116),DEF=c(60,51,85),
Sp.A=c(80,174,110),Sp.D=c(80,285,111),SPD=c(50,122,189)),
EV=data.frame(EVHP=c(0,0,50),EVATK=c(0,0,50),EVDEF=c(0,0,50),
EVSp.A=c(0,0,50),EVSp.D=c(0,0,50),EVSPD=c(0,0,50)),
                     lvl=c(50,100,75),
                     nature=c("Brave","Brave","Hasty"),
                     interesse=3)
 ## Exemplo simulando muitos pokémon
  a = matrix(round(runif(300,0,31)), ncol=6)
  b = data.frame(rep("Articuno",50))
  base.stats =
read.csv(url("http://ecologia.ib.usp.br/bie5782/lib/exe/fetch.php?media=bie5
782:01_curso_atual:alunos:trabalho_final:diego.pereira.silva:base.stats.csv"
```

```
), as.is=T)
  nat.mat = matrix(c(rep(1,
5),1.1,0.9,1,1,1,1.1,1,0.9,1,1,1.1,1,1,0.9,1,1.1,1,1,1,0.9,0.9,1.1,1,1,1,rep
(1, 5),
1,1.1,0.9,1,1,1,1.1,1,0.9,1,1,1.1,1,1,0.9,0.9,1,1.1,1,1,1,0.9,1.1,1,1,rep(1,
5),1,1,1.1,0.9,1,
1,1,1.1,1,0.9,0.9,1,1,1.1,1,1,0.9,1,1.1,1,1,1,0.9,1.1,1,rep(1,
5),1,1,1,1.1,0.9,0.9,1,1,1,1.1,
                        1,0.9,1,1,1.1,1,1,0.9,1,1.1,1,1,1,0.9,1.1,rep(1,
5)),
                    ncol=5, byrow=T,
dimnames=list(c("Hardy","Lonely","Adamant","Naughty","Brave","Bold","Docile"
,"Impish","Lax","Relaxed",
"Modest","Mild","Bashful","Rash","Quiet","Calm","Gentle","Careful","Quirky",
"Sassy",
"Timid","Hasty","Jolly","Naive","Serious"),
                                    c("ATK","DEF","Sp.A","Sp.D","SPD")))
  b[,2:7] = (c(round((((2*base.stats[base.stats[,1]=="Articuno",2] + a[,1] +
0 + 100) * 100) / 100 + 10)),
               round((((2*base.stats[base.stats[,1]=="Articuno",3] + a[,2] +
0) * 100) / 100 + 5) * nat.mat["Calm",1]),
               round((((2*base.stats[base.stats[,1]=="Articuno",4] + a[,3] +
0) * 100) / 100 + 5) * nat.mat["Calm",2]),
               round((((2*base.stats[base.stats[,1]=="Articuno",5] + a[,4] +
0) * 100) / 100 + 5) * nat.mat["Calm",3]),
               round((((2*base.stats[base.stats[,1]=="Articuno",6] + a[,5] +
0) * 100) / 100 + 5) * nat.mat["Calm",4]),
               round((((2*base.stats[base.stats[,1]=="Articuno",7] + a[,6] +
0) * 100) / 100 + 5) * nat.mat["Calm",5])))
  b[,8:13] = 0
  b[,14] = 100
  b[,15] = "Calm"
  exemplo4 = iv.calc(b[,1], b[,2:7], b[,8:13], b[,14], b[,15], interesse=25)
```
