Silas Principe

Doutorando em Oceanografia (Oceanografia Biológica), Instituto Oceanográfico, USP

Minha pesquisa visa modelar a distribuição de espécies de ambientes costeiros rasos em alguns cenários climáticos futuros para entender como as mudanças climáticas poderão afetar essas espécies e as comunidades associadas.

Exercícios

Exercícios

Trabalho final

Proposta A: visualR

Função para criar diagramas

Diagramas e mapas conceituais são ferramentas úteis para a melhor compreensão de questões científicas. O raciocínio lógico é consideravelmente ampliado quando diagramas são utilizados [1]. Na área da saúde, por exemplo, o uso de diagramas diretos acíclicos (DAG, Direct Acyclic Graphs) é utilizado para verificar variáveis para as quais é necessário um ajuste na análise [2].
A proposta é criar uma função que permita ao usuário criar diagramas/mapas conceituais de maneira manual, alem de dar a opção de criar um DAG automaticamente. A ideia para essa proposta surgiu após um amigo, que cursa pós em nutrição, me procurar para entender alguns pontos de um artigo. Quando me deparei com o DAG vi as potencialidades de seu uso e resolvi tentar criar essa função.

Plano da função:

Entrada: visualR(text, color, line, words, arrows, rectangle, font)

`text` = vetor com as palavras/números que devem ser inseridos no diagrama. Todos os valores serão convertidos para a classe “character”. Opcional. (classe: character, numeric ou factor)
`color` = opcional, define a cor dos retângulos e setas do diagrama (uso com palavras como “red”, “blue”, ou RGB). (classe: character)
`line` = opcional, por padrão é do tipo 2. Define o tipo de seta que será usado ou se o usuário usará apenas linhas (será definido no help). (classe: numeric)
`words` = opcional caso `text` não seja definido. Delimita quantas palavras o usuário poderá inserir. Caso `text` esteja presente, é ignorado. Se `text` e `words` não for especificado, o usuário terá a opção de digitar até 200 palavras, com opção de parar quando desejar. (classe: numeric)

`arrows` = opcional caso `text` não seja definido. Delimita quantas setas ou linhas o usuário poderá inserir. Se `text` e `words` não for especificado, o usuário terá a opção de desenhar até 200 setas/linhas, com opção de parar quando desejar. (classe: numeric)
`rectangle` = por padrão é TRUE. Caso o usuário não deseje retângulos ao redor do texto, pode optar por FALSE.
`font` = permite ao usuário escolher o tipo de fonte que será usado (especificado no help da função `par`, em `family`).

Verificando os parâmetros `text` é do tipo character? Se não, converte para character. Verifica o tamanho do vetor. O vetor tem comprimento maior do que dois? Se não, para a função e escreve "TEXT deve ser um vetor com comprimento maior do que dois".
`words` e `text` estão presentes juntos? Se sim, escreve "WORDS não é um argumento quando TEXT está presente. Ele será ignorado." e continua a função
Se `text` está ausente e `words` também, imprime a mensagem: "WORDS e TEXT não especificados. O usuário terá um limite de 200 palavras, podendo parar quando quiser."
`arrows` é um valor maior do que 0? Se não, para a função e escreve "ARROWS deve ser um valor maior do que 0".

Pseudo-código

- Cria um espaço gráfico com configurações padrões
- Se `text` está presente

1. Cria o objeto `qt` com o comprimento de `text`

- Se `text` não está presente, mas `words` está

1. Insere o valor de `words` em um objeto `qt`

- Se `text` e `words` não estão presentes

1. Cria objeto `qt` com valor 200

- Plota um gráfico de 0:50 nos dois eixos, sem nenhum tipo de informação (espaço em branco)
- Se `text` e `words` não estão presentes

1. Usando for de 1:qt
    1. Imprime a mensagem "onde você quer o texto?"
    2. Cria o vetor `point` com locator(1) (usuário clica onde quer o texto no gráfico)
    3. Apresenta a mensagem "Escreva seu texto (max 10 caracteres) ou digite END para encerrar a digitação de textos." (dentro da função readline)
    4. Cria o vetor `text.r` com o texto inserido pelo usuário usando a função readline
        1. Se text.r for END, encerra a digitação de textos e vai para a inserção de linhas. Se não:
            1. Verifica se o texto inserido está dentro do máximo permitido de caracteres (10, se não ocupa muito espaço)
            2. Se não, mostra mensagem avisando que o texto é maior que o máximo permitido e pede para o usuário escrever de novo
            3. Se novamente o texto estiver maior que 10 caracteres, para a função e exibe mensagem de erro
    5. Se `rectangle` é TRUE

1. Cria os objetos `text.w` e `text.h` com o comprimento e a altura, respectivamente, de `text.r`
2. Cria um vetor x0 com o valor de x em `point - sw`
3. Cria um vetor x1 com o valor de x em `point + sw`
4. Cria um vetor y0 com o valor de y em `point - sh`
5. Cria um vetor y1 com o valor de y em `point + sh`
6. Cria um retangulo com x0, y0, x1, y1 e cor rgb pegando a posição i de cada coluna de colors.

6. Adiciona o texto da posição i de `text` na posição x e y de `point`
7. Usando for de 1:qt (ou arrows, se `arrows` estiver presente)
    1. Imprime a mensagem “onde você quer a flecha?”
    2. Cria o vetor `point` com locator(2)
    3. Cria um objeto `point` com os valores de locator(2) (usuário deverá clicar no gráfico duas vezes, ponto inicial e final da linha)
    4. Gera a linha com a posição x1, x2, y1, y2 de `point`, parâmetro `line` e cor `colors`.
    5. Pergunta se o usuário quer inserir outra linha (usando readline). Se não, usa `return` para encerrar o diagrama.

- Se `text` ou `words` estão presentes

1. Usando um for de 1:qt
    1. Cria um objeto `point` com o valor de locator(1) (usuário deverá clicar no gráfico uma vez para cada palavra)
    2. Se text não está presente
        1. Cria o vetor `text.r` com o texto inserido pelo usuário usando a função readline
        2. Verifica se o texto inserido está dentro do máximo permitido de caracteres (10, se não ocupa muito espaço)
        3. Se não, mostra mensagem avisando que o texto é maior que o máximo permitido e pede para o usuário escrever de novo
        4. Se novamente o texto estiver maior que 10 caracteres, para a função e exibe mensagem de erro
    3. Se `rectangle` é TRUE
        1. Se text está presente
            1. Cria os objetos `text.w` e `text.h` com o comprimento e a altura, respectivamente, do texto na posição i de `text`
        2. Se text não está presente
            1. Cria os objetos `text.w` e `text.h` com o comprimento e a altura, respectivamente, de `text.r`
        3. Cria um vetor x0 com o valor de x em `point - sw`
        4. Cria um vetor x1 com o valor de x em `point + sw`
        5. Cria um vetor y0 com o valor de y em `point - sh`
        6. Cria um vetor y1 com o valor de y em `point + sh`
        7. Cria um retangulo com x0, y0, x1, y1 e cor `colors`.
    4. Se text está presente
        1. Adiciona o texto da posição i de `text` na posição x e y de `point`
    5. Se text não está presente
        1. Adiciona o texto de text.r na posição x e y de `point`
2. Usando um for de 1:qt (ou de 1:arrow, caso `arrow` esteja presente).
    1. Cria um objeto `point` com os valores de locator(2) (usuário deverá clicar no gráfico duas vezes, ponto inicial e final da linha)
    2. Gera a linha com a posição x1, x2, y1, y2 de `point`, parâmetro `line` e cor `colors`.

Saída

- Retorna um objeto gráfico com o diagrama escolhido.

> — *Renan Del Bel 2019/06/13 12:53*
>
> Toda a parte relacionada a DAG é muito confusa, será mais trabalhosa do que o adequado para disciplina e precisa de revisão.
>
> Entretanto, a parte ‘free’ está clara e provavelmente será o suficiente para seu trabalho final.
>
> Se seguir nessa proposta, recomendo trabalhar apenas como 'free' e dar mais opções ao usuário. Por exemplo a opção de escolher se o diagrama vai ou não ter setas e se vai ou não ter o retângulo ao redor da palavra.
>
> A opção de iniciar a função sem saber quantas palavras e setas irá usar também seria interessante, para isso basta criar uma forma do usuário indicar que terminou de desenhar (por exemplo, depois de cada palavra/seta perguntar se já terminou ou talvez se clicar fora da área do gráfico)
>
> Considere fazer com que o tamanho da área do plot seja customizável ou dependa de quantas palavras o usuário pretende por.
>
> O argumento text precisa mesmo ser character? O nome dos nós de um diagrama podem ser números e é possível que o usuário envie fatores para usar como nome. Ao invés de parar a função, vale a pena converter o input do usuário em character e depois verificar se a conversão foi bem sucedida.
>
> Também não há a necessidade de parar a função porque o usuário enviou input que não será usado, basta emitir um aviso.
>
> O passo 6 e 7 são idênticos, não há porque separá-los, coloque a verificação da presença de 'text' dentro do seu loop.
>
> No passo 7.III.B, em vez de parar a função, basta pedir um input novo.

> — *Silas Principe 2019/06/19 10:02*
>
> Renan, segue versão atualizada do projeto. Seguindo suas recomendações, utilizarei apenas a proposta do diagrama 'free'. Também alterei a escolha de cores, e dei a opção de ter ou não o retângulo. Agora há a opção de escolher o tipo de fonte.
>
> Não achei conveniente dar a opção de não inserir nenhum tipo de linha.
>
> Agora também há a opção de não inserir um vetor `text` e não especificar `words`.

Nesse caso o usuário terá um limite de 200 palavras, podendo parar quando desejar.

`text` também pode ser um vetor numérico ou de fatores, sendo convertido para character na função.

— Renan Del Bel 2019/06/19 12:53

Está muito melhor! Tenho algumas pequenas sugestões, mas já está muito bom.

A parte que roda quando text está presente e a parte que roda quando não está presente são praticamente idênticas. Isso é um bom indicativo que elas não precisam estar separadas.

Ao invés de fazer um IF no começo da função que separa ela em dois caminhos totalmente independentes, pense no que acontece de diferente em cada opção e tente colocar IFs apenas para esses casos.

Um pequeno detalhe: END é uma palavra razoavelmente comum para se colocar em um fluxograma. Eu pensaria em um sinal de parada que inclui caracteres especiais, apenas para ter menos chance de um usuário se frustrar, algo como /END ou <END>

Por fim, se você vai por um limite de 10 caracteres verifique antes se “text” cumpre esse requisito. Seria chato se a função parasse no meio, depois de você cuidadosamente posicionar a umas 15 palavras, porque seu input não estava adequado desde o inicio.

Acho que é isso!

Proposta B: eatR

Função para cálculo de receitas

O desperdício de alimentos é um problema global. Dados da FAO indicam que 1/3 de todo o alimento produzido no mundo é perdido ou vai para o lixo[3]. Uma das formas de evitar o desperdício é cozinhar apenas o necessário para a refeição em questão. No entanto, aqueles que tem afinidade com a gastronomia sabem a dificuldade de reduzir ou aumentar receitas. Normalmente apenas se divide ou multiplica a receita original por dois, uma vez que encontrar valores intermediários é um exercício demorado. Assim, o objetivo dessa proposta é criar uma função que facilite o cálculo de receitas para uma quantidade *n* de pessoas considerando um tamanho de porção individual adequado para aquela receita. O resultado final é um *dataframe* contendo os ingredientes com as quantidades originais, o valor convertido e, no caso de receitas com ovos, um valor ajustado (uma vez que seria mais desperdício usar meio ovo).

Plano da função:

Entrada: eatR(rec, recipe, persons, portion, egg.w=60)

`rec` = dataframe com duas colunas contendo 1) ingredientes (ing); 2) quantidades (qtt). No caso de um dos ingredientes ser ovo, o nome deve ser escrito em minúsculo, em português ou inglês, no singular. (classe: dataframe)
`recipe` = por padrão é NULL. Caso o usuário decida usar uma das receitas pré-definidas na função. Serão 4 opções a serem descritas no help. Ex: “pizza”. (classe: character)
`persons` = quantidade de pessoas para as quais a receita deve ser calculada. (classe: numeric)
`portion` = valor da porção individual. O usuário deve estimar a quantidade que uma pessoa come daquela receita. (classe: numeric)
`egg.w` = opcional, por padrão é 60. Permite ao usuário definir, nos casos em que a receita apresenta ovo, qual o peso do ovo (classe: numeric).

Verificando os parâmetros

`recipe` está presente? Se sim, faz parte de uma das quatro receitas disponíveis? Se não, escreve “RECIPE deve ser uma das receitas pré-definidas.”
`rec` está presente*? Se não, escreve “REC é necessário para o cálculo da receita”. (*pode ser não caso `recipe` esteja presente). É um dataframe contendo uma coluna de nome “ing” (ingredientes) e outra “qtt” (quantidade)? Se não, escreve “REC deve ser um dataframe com duas colunas, uma de ingredientes (ing) e outra com as quantidades (qtt)”.
`persons` está presente e é da classe numérica? Se sim, é maior ou igual a 1? Se não, escreve “PERSONS deve ser um número maior ou igual a 1”.
`portion` está presente e é da classe numérica? Se não, escreve “PORTION deve ser um número”. Adicionalmente, faz um teste para ver se o valor inserido é menor que 10 (uma porção improvável). Nesse caso apenas avisa “PORTION é um valor muito pequeno. Talvez necessite de revisão.”
`egg.w`: verifica se o valor é menor que 40 ou maior que 70[4] e apenas avisa “EGG.W muito pequeno/grande. Talvez necessite de revisão.”

Pseudo-código

- Para caso `recipe` seja utilizado
- Cria quatro dataframes, cada um com uma das receitas pré-definidas (nome dos dataframes == nome da receita)
- Cria lista `recipes` com os quatro vetores
- Substitui objeto `rec` pela receita utilizada indexando a lista com o objeto `recipe` usando o código `(recipes[ [as.name(recipe)] ]`
- Se ovo estiver presente na receita

1. multiplica o a quantidade de ovo na coluna `qtt` do objeto `rec` pelo valor do objeto `egg.w`
2. substitui o valor de ovo na coluna `qtt` pelo valor calculado

- Cria o objeto `sum.tot` com a soma do peso total da receita original somando os valores de `qtt`
- Cria o objeto `por.tot` com o valor arredondado da divisão de `sum.tot` por `portion`
- Cria a coluna `conv` no dataframe `rec` com os valores da coluna `qtt` multiplicados por `persons` e dividido por `por.tot`
- Se ovo estiver presente na receita

1. Cria o objeto `eggs` com o valor de ovo na coluna `qtt` e divide por `egg.w` para retornar em 'número de ovos'
2. Cria o objeto `e.round` com o valor aproximado para o próximo inteiro (usando a função ceiling)

de `eggs` dividido por `eggs`. Isso dará um valor de correção
3. Cria a coluna `adj` no objeto `rec` com os valores da coluna `conv` multiplicados por `e.round`
4. Substitui o valor de ovo na coluna `adj` pelo valor aproximado para o próximo inteiro (usando a função ceiling) de `eggs`
5. Transforma (e substitui) os valores de ovo nas colunas `qtt` e `conv` dividindo o valor `por egg.w` (retornando assim ao valor em número de ovos, e não peso)
6. Arredonda os valores da coluna `adj` do objeto `rec`

- Arredonda os valores da coluna `conv` do objeto `rec`
- Imprime mensagem na tela com “Receita final:“

Saída

- Retorna um dataframe com as colunas `ing`, `qtt`, `conv`, e caso tenha ovos `adj`

> *— Renan Del Bel 2019/06/13 12:53*
>
> Achei a ideia interessante, tenho algumas sugestões:
>
> Em primeiro lugar, em vez de incluir receitas prontas na sua função, deixe para usá-las no exemplo do help.
>
> Exigir que o data frame que você recebe tenha os nomes certos nas colunas não é necessário, já que você pode atribuir nomes se quiser ou trabalhar apenas com a indexação.
>
> Além disso, pense sobre como receitas normalmente são apresentadas:
>
> - Raramente todos os ingredientes têm a mesma unidade de medida, então somar esses valores e dividir por 'portion' não é interessante.
> - Por outro lado, muitas vezes é informado quantas porções ela serve. Então os argumentos 'persons' e 'portion' podem ser mudados para 'quantidade que a receita original' faz e 'quantidade que quero fazer'.
>
> Sobre os ovos: Não entendi direito como o cálculo foi feito: eggs/eggs será sempre igual a 1, não?
>
> Sugiro que deixe qtt de ovos ser em unidades. Com isso você já sabe quantos ovos por porção a pessoa vai usar e então não precisa mais de eggs.w.
>
> Mas eu permitiria deixar a pessoa escolher como arredondar o ovo. Talvez ela prefira diminuir a receita caso 'conv' tenha calculado que precisa de 6.1 ovos ou coisa assim.

> — *Renan Del Bel 2019/06/13 12:53*
>
> [editado: *2019/06/17 12:53*]
>
> Recomendo seguir com a proposta A, mas fazendo as mudanças indicadas. Tente reescrever o pseudo-código e colocar aqui embaixo (pode ser menos detalhado dessa vez, apenas para fecharmos o escopo de seu trabalho).
>
> Lembrando que o objetivo é que a função seja um desafio, então vamos conversando para acertar o nível de complexidade adequado.
>
> Casa tenha alguma dúvida ou comentário, pode entrar em contato por e-mail (delbel.renan@gmail.com) ou colocar aqui na página mesmo.

Referências

[1] Bauer, M. I., & Johnson-Laird, P. N. (1993). How Diagrams Can Improve Reasoning. Psychological Science, 4(6), 372–378. https://doi.org/10.1111/j.1467-9280.1993.tb00584.x
[2] Marit M. Suttorp, Bob Siegerink, Kitty J. Jager, Carmine Zoccali, Friedo W. Dekker, Graphical presentation of confounding in directed acyclic graphs, Nephrology Dialysis Transplantation, Volume 30, Issue 9, September 2015, Pages 1418–1423, https://doi.org/10.1093/ndt/gfu325
[3] http://www.fao.org/food-loss-and-food-waste/en/
[4] http://www.uniquimica.com/2016/11/classificacao-de-ovos/

Resultado final

Proposta A

Página com o código da proposta A: visualR

Página com o help da função: Help visualR