**Disciplina Python**

Ideia: criar uma forma de lógica; desenvolver raciocínio para análises

Entender análise, consegue fazer qualquer uma.

Não ficar preso a um pacote

Notas: trabalho de conclusão (0.7) + trabalho em aula (0.2); participação (0.1)

Prazo: 1 dia antes do último dia de aula

**Trabalho final (30/05): paper + repositório do github**

- gerar tabela de OTU (cada linha uma espécie, cada coluna uma amostra): 26 colunas e 100 linhas (linhas nomeadas como OTU\_numero)

- cada amostra tem que ter entre 40 a 70% de zeros; tem que somar até 100.000

- OTU table é como o resultado de uma análise.

Depois da tabela: rarefação -> um tipo de normalização dos dados; pego o mínimo de reads dentro de todas as minhas amostras (ex:30.000) e tiro elementos até ficarem todos com esse número de reads

Ideia: escrever função para rarefar a tabela

Depois outro tipo de normalização: vou pegar cada coluna e dividir pelo total de cada coluna – vou ter a proporção de cada sp em cada amostra -> aí vai multiplicar por um numero mtt grande (ex 100000) => TSS

Ideia: encontrar referências para dizer qual é melhor e usar (rarefação ou TSS) nossos dados para explicar.

Próx: fazer boxplot para as médias de contagem para cada tratamento, um pra tabela original, um pra rarefeito e um TSS para comparar

Calcular as médias e desvios padrão

Depois: curva do coletor – um gráfico onde tenho o tamanho da amostra pelo n de espécies observadas; em cada ponto tem um tamanho amostral -> criar função para ter essa curva do coletor razoável (tem que ter mts amostragens)

Por fim: calcular função do índice de shannon

**Para dia 17: resumo do PEP8**

**Documentos com as instruções por e-mail**

Usuário = conhece a ferramenta

Programador = sabe escrever, achar erro, questionar

Programa / software= conjunto de funções que roda em cima de uma linguagem

Script = todo script é um programa;

R é uma linguagem, não programa.

PYTHON

É uma linguagem interpretativa; interpreter converte o que escrevi em binário para a máquina executar;

Vantagens: gerenciamento automático de memória; expressividade e sintaxe em inglês; facilidade de programação; minimiza o tempo de desenvolvimento; foco na importação de módulos (útil para computação científica) – pode subir na nuvem do python ou chamar de qualquer lugar

Sugestão de pacotes com linguagem basal: itertools (rápido); collections (pouca memória); gzipi lzma

Github – repositório, você coloca lá, outras pessoas rodam e podem falar onde teve erro

Consegue fazer rastreabilidade das mudanças

Lógica = padrões, semântica, usa regras

Estilos de programação = imperativo, funcional, lógico, orientado. Aqui vamos aprender o lógico.

Diferença dos estilos é o nível de abstração

Lógica – forma básica pela qual os programadores entendem e organizam seu código

Segue: sequência; seleção das instruções (if-else); iteração (loops, um passo repetido)

**Operadores: Python tem vários**

Operadores de assinatura (assigment): consegue adicionar operações (soma, divisão, resto...) com sinal de = (o sinal é o operador de assinatura)

Operador de membership: está dentro ou não (in or not in)

Operadores lógicos:

And = dá true quando duas condições são verdadeiras

Or = só dá falso quando as duas são falsas

Not = reverte; not false = true

Not serve como um controle de processo - posso usar “if not [x] ...”

Se a função não correr no python aparece “none”

Operadores de identidade: se uma coisa é aquilo ou não; is e is not

Operadores de comparação: == (equal to), != (not equal to), < (less than), > (more than), <= (less than or equal to), >= (more than or equal to)

Operadores aritiméticos: unário (deixa numero negativo ou positivo), +, -, \*, /, % (modulo), // (floor Division), \*\* (exponenciação)

1e-3 = 10-3

Ordem das operações: parênteses, exponenciais, multiplicação/divisão, adição/subtração -> seguir colocando parênteses, separar operações para facilitar

Precedência das operações também segue lógica: \*\* -> \* / % // -> + - -> == != <= = < > -> is is not -> not

Exercícios:

1. Verificação maior ou menor de idade

Input (x)

If x >= 18:

Maior de idade

Else: menor de idade

1. Dado dois números, retorne qual deles é maior ou se ambos são iguais

Input (x)

Input (y)

If x>y:

“X”

If y>x:

“Y”

Else:

“x=y”

1. Dado uma nota, indique qual o conceito do aluno

Input (x)

If x >= 9:

“a”

If X>= 8:

“b”

If X >= 5:  
“c”

Else:

“d”

1. Dada uma lista de números, calcule a soma dos números pares (para próxima aula)

Algoritmos = você tem uma entrada, um processo (o algoritmo) e uma saída

Ex: organização – algoritmo de sort – processo de organização, tem várias formas de fazer

Características: entrada; saída; definição; te dá um resultado correto; é finito; efetividade (passos individuais que quando juntos são factíveis); generalização e eficiência (toma tempo e memória pequeno p realizar)

Tempo de operação unitária = n de operações que um algoritmo realiza. Quanto mais complexo, mais demorado.

Processo mais rápido: constante O(1) = não depende do meu input ou processamento. Não é um processo perse.

Se depende de um processamento uma única vez = O(n); se depende 3 = if...if..if... = O(n^3)

De linear para baixo é um bom algoritmo – linear O(n); logaritmo O(log(n))

A complexidade determina o tempo:

Para cada amostra tiro 20 subamostras, para cada subamostra faço 10 replicatas e tiro média dessas replicatas

1 \* 20 \* 10 \* 1 = 200 -> para cada amostra saio com 200, assim o processo cresce quase exponencial

Se tenho 100000 reads por amostra:

De mil em mil, dez replicatas, 26 amostras = a computação vai crescer 260 000 -> vamos ter que fazer 260 mil computações

Estrutura – algoritmos tem que ser escritos de forma científica

No algoritmo, setinha é igual (é uma linguagem antiga, não existia igual ainda)

Para calcular media:

Input: lista de números positivos

Output: numero positivo

M <- 0 ; L<- 0

Para cada X in lista do:

M+= x

L +=1

End

Return m/L

**Exercícios:**

Calcular o índice de Shannon = é uma medida de entropia (n de estados possíveis que o sistema pode adquirir, ela é maior quanto menos homogêneo o sistema for) -> não pode ser negativo nunca.

Como calcula: somatório da proporção x ln proporção

Fazendo por amostra (uma coluna da nossa lista):

Algoritmo: calculo do índice de Shannon

Input: uma lista de valores inteiros positivos

Output: um valor positivo

M = 0 ; new\_list [] ; H = 0

For X in list do:

M += x

For X in list do:

New\_list << x / m

For x in new\_list do:

X <- x \*ln x

For x in new\_list do:

H += x

End

Return -1\*H

Outra forma: (mas não é generalizável, então melhor com “X”)

Total = 0 ; newlist = []

Para abundancia em lista:

Total += abundancia

Para abundancia em lista?

NL = abundancia / total

**Curva do coletor: relaciona n de novas sps encontradas com o aumento do numero de amostras**

Tenho 1000 reads -> tiro 100 amostras -> conto sps

1ª amostra aleatória = encontrei 10 sps ; 2ª somo com a primeira = encontrei 15 sps ... e assim vai

Problema: se eu faço uma única vez, a curva do coletor fica estranha

Segunda forma: posso fazer retirando amostras maiores (por ex mil, depois dois mil, etc).

Caminho várias subamostras:

Algoritmo: curva do coletor

Input: lista de abundancia de sps, numero de subamostras, pseudorréplicas

Output: lista com numero de reads e numero de espécies médio observado

Primeiro calcula o total de reads por subamostra

Por ex: tenho 100000 reads, quero subamostras de 100 reads, então 100 000/100 = 1000.fazendo:

Total = 0 ; L = [ ]

For x in list\_ab:

Total += 0

Tamanho <- (total / n subamostras)

Para cada subamostra e replica:

For each réplica:

L = [ ];

a = choose (tamanho; lista)

L << a

Agora vamos contar: 1º total, 2º tamanho da subamostra; 3º pra cada replica, amostrar aleatoriamente x elementos (x = tamanho da subamostra); 3.1º contar o número de espécies; 3.2 para cada subamostra, somo e conto o número de espécies (não conto o que já saiu; até totalizar, é a ideia da curva); para fazer a curva: somar elemento a elemento e dividir pelo numero de réplicas. Então, o resultado do algoritimo tem que ser uma lista com 2 valores para construção da curva.

Fazer em casa

Turnover: dado um conjunto de amostras, quero saber o quanto varia dentro delas.

É o calculo do diagrama de Venn; Jacard = o que tem de igual em A e B / total

Nestedness = 1 – Jacard ; turover = Jacard

Se somar as duas, dá 1

Ex: 5 em A; 3 em B; 10 em A e B

J = 10 / 10+5+3 => diz o quão similar um é do outro

Algoritmo: turnover

Input: amostra 1 e amostra 2

Output: valor decimal positivo representativo do turnover

Criar listas separadas para cada e uma com a união

a []; b = [ ]; ab = []

For sp in amostra 1 do:

If sp in amostra 2:

ab << sp

else:

a << sp

For sp in amostra 2 do:

If sp in amostra 1:

ab << sp

else:

b << sp

A = 0 ; B = 0 ; AB = 0

Para calcular o tamanho:

A = len(a)

B = len (b)

AB = len (ab)

Para saber o total de sp observadas:

T = A + B + AB

End

Return 1-AB/T

Dá pra calcular turnover com Bray-curtis também, que leva abundancia em questão. Poucas sp = geralmente usa Bray-curtis; se for 5 mil ou mais sp você não consegue calcular com Bray-curtis, não é mais informativo

**Rarefação:**

Deixa tudo na mesma quantidade de reads da menor amostra que temos

Não pode subamostrar aleatoriamente pq perde a proporção original das sps

Vamos escrever resumido porque o convencional é grande. 1º = saber qual é minha menor amostra (somar as linhas para ter o total por colunas); 2 = achar o valor mínimo das somas; 3 = calcular a proporção de cada sp em cada amostra 5= procedimento de amostragem aleatória para preencher as artificiais, mas levando em conta a proporção das espécies;

Rarefação não é a mais eficiente pq não necessariamente mantem as proporções – existem métodos mais simples que mantem a proporção -> se mantém proporção e diminui n de reads, vc pode excluir espécies raras.

**TSS:** É a rarefação até a proporção, mas aí não faz o mínimo, mas multiplica por um numero muito grande para não perder espécies; depois disso retorna em uma matriz de inteiros

1. Somar todas as linhas de cada coluna (ver n total de reads);
2. proporção de cada espécie dentro de cada amostra (divide abundancia pelo total de sp da amostra);
3. para restituir de proporção a número inteiro (integro) tem que multiplicar por um numero muito grande a tabela toda (faz a mesma coisa que a rerefação, mas sem aleatoriedade e logo sem perder sp raras), que pode ser o valor máximo da etapa 1;
4. Fazer com que os números se tornem inteiros (usa função int); tabela normalizada com TSS

Como determinar o número muito grande que você multiplica? => pode usar o valor da amostra com máximo das reads.

Exercícios: 1- determinar raiz quadrada; 2- arredondar 1 número; 3- sorting lista de valores positivos; 4- determinar quociente e resto de divisão

Algoritmo: determinar raiz quadrada

Input: número positivo

Output: raiz quadrada do numero

n\*\*(1/2)

Se não tivesse o exponencial: o que faria? -> divide pela metade, eleva ao quadrado, se ficar mt longe pega a metade e eleva... assim vai até chegar mais próximo. Posso colocar uma condição: que a diferença máxima seja tal.

Algoritmo: arredondar números

Input: número real com numero de casas decimais

Output: numero com as casas decimais escolhidas

Contar numero de casas decimais de interesse; olhar a regra; retornar o numero.

3

Algoritmo: Sorting lista de números positivos

Input: lista de números

Output: organização números positivos

Uma forma: calcula máximo e mínimo, depois criar ranges e organizar dentro de cada range. Problema: tamanho dos rangers, tamanhos diferentes.

Outra forma: pega um valor no meio e vai comparando, é maior ou menor? Vai pra frente ou pra tras dele

Outra forma: corto a lista em duas e jogo pra lá e pra cá; se é maior troco, se é menor fica.

Outra forma: pego cada elemento e vou comparando com o anterior. Problema: em uma comparação faz várias, é demorado.

4

Algoritmo: determinar o quociente e o resto de uma divisão

Input: número divisor (d), número dividendo (D)

Output: quociente, resto

N = [ ]

For x from 1 at infinito:

R= D – (x d)

Mas tem as funções // e %

Tarefa: pegar um problema de escolha pessoal e criar um algoritmo para resolver ele, de preferencia com dados.