Alunas: Anna Carolina Silva Garcia e Beatriz Pereira da Silva e Souza

Projeto Final

O objetivo deste projeto é aplicar todos os conhecimentos adquiridos na disciplina de programação para biociências e disciplinas de pré-requisito para resolver um problema cotidiano da área de bioinformática.

LINK PARA O CÓDIGO: GitHub - CFB017 - Projeto Final

Caso: Foi realizado o sequenciamento de RNA de um organismo com parentesco com o *Rhodnius prolixus* chamado de *Rhodnius desconhecidus*. Queremos saber qual a natureza dos genes mais expressos em cada condição. No entanto, o *R. desconhecidus* foi recentemente sequenciado e não possui ainda a descrição completa do produto de cada gene anotado. Portanto, devemos buscar por genes do seu parente próximo que tenham homologia baseando-se na similaridade por sequência. Para resolver este problema, desenvolva um código em Python que atenda os critérios e realize as funções seguintes:

A) Leia da linha de comando nesta ordem:
   I) uma tabela xlsx com dados de quantificação (Tabela 1) para quatro bibliotecas de RNA-Seq;
   II) Um arquivo multi-FASTA contendo as sequências de DNA dos genes de *R. desconhecidus* (Arquivo 1);
   III) Um arquivo multi-FASTA contendo sequências de aminoácidos de *R. prolixus* (Arquivo 2);

A Tabela 1 contém os identificadores dos genes e os valores de expressão não-normalizados para cada uma das duas réplicas (Rep1 e Rep2) para cada condição (A e B);

B) Crie colunas adicionais para adicionar os níveis de expressão normalizados por CPM (counts per million) de cada réplica. Nomeie essas colunas como Rep1_A_CPM, Rep2_A_CPM, Rep1_B_CPM e Rep2_B_CPM;

C) Crie colunas adicionais que vão armazenar a expressão normalizada (CPM) média por condição. Nomeie as colunas como Cond_A_CPM_media e Cond_B_CPM_media;

D) Selecione os cinco genes mais expressos de cada condição baseado na expressão média.

E) Realize uma busca BLAST da sequência de DNA dos 10 genes selecionados anteriormente contra as sequências de aminoácidos de *R. prolixus*.

F) A partir do resultado do BLAST, imprima o melhor hit para cada um dos 10 genes baseado no maior valor de bitscore. Em caso de bitscores iguais, selecione o hit com menor e-value. Caso o empate persista, selecione qualquer um dos hits empatados.

G) A saída impressa no terminal deverá conter os seguintes dados:

gene_id	Cond_A_CPM_media	Cond_B_CPM_media	id_proteína_encontrada

Onde gene_id é um dos genes mais expressos na condição A ou B; Cond_A_CPM_media e Cond_B_CPM_media são os valores médios de CPM para cada condição; e id_proteína_encontrada é o identificador da sequência de proteína com melhor hit em *R. prolixus.*

Alunas: Anna Carolina Silva Garcia e Beatriz Pereira da Silva e Souza

Código Final:

```
# importação de bibliotecas:
import sys
import pandas
import pandas as pd
import xlrd
from Bio.Seq import Seq
from Bio import SeqIO
from Bio.SeqRecord import SeqRecord
from Bio.Blast.Applications import NcbiblastxCommandline
# blastx path:
blastx = "/home/carol/anaconda3/bin/blastx"

## ITEM A:
#interação com o termiunal/linha de comando:
tabela = sys.argv[1]
arquivo1 = sys.argv[2]
arquivo2 = sys.argv[3]
RNA_seq = pandas.read_excel(tabela)
Rhodnius_desconhecido = SeqIO.parse(open(arquivo1,'r'), "fasta")
Vector_Base =  SeqIO.parse(open(arquivo2,'r'), "fasta")
Rhodnius_data = arquivo2

# para a IDE:
#     RNA_seq     =     pandas.read_excel('/home/carol/Documents/CFB017/Projeto
Final/Tabela_1.xlsx')
#                              Rhodnius_desconhecido                                =
SeqIO.parse(open("/home/carol/Documents/CFB017/Projeto
Final/Rdesconhecidus.fasta",'r'), "fasta")
#   Vector_Base   =      SeqIO.parse(open("/home/carol/Documents/CFB017/Projeto
Final/VectorBase-48_RprolixusCDC_AnnotatedProteins.fasta",'r'), "fasta")
#  Rhodnius_data  =  r"/home/carol/Documents/CFB017/Projeto  Final/VectorBase-
48_RprolixusCDC_AnnotatedProteins.fasta"

## ITEM B:
# adiciona novas colunas à tabela.
# os valores dessa coluna serão as quantidades absolutas calculadas com a
fórmula de CPM.
RNA_seq['Rep1_A_CPM'] = RNA_seq['Rep1_A']/(10**6)*(RNA_seq['Rep1_A'].sum())
RNA_seq['Rep1_B_CPM'] = RNA_seq['Rep1_B']/(10**6)*(RNA_seq['Rep1_B'].sum())
RNA_seq['Rep2_A_CPM'] = RNA_seq['Rep2_A']/(10**6)*(RNA_seq['Rep2_A'].sum())
RNA_seq['Rep2_B_CPM'] = RNA_seq['Rep2_B']/(10**6)*(RNA_seq['Rep2_B'].sum())

## ITEM C:
# calculo da média:
RNA_seq['Cond_A_CPM_media']          =          (RNA_seq['Rep1_A_CPM']          +
RNA_seq['Rep2_A_CPM'])/2
RNA_seq['Cond_B_CPM_media']          =          (RNA_seq['Rep1_B_CPM']          +
RNA_seq['Rep2_B_CPM'])/2
```

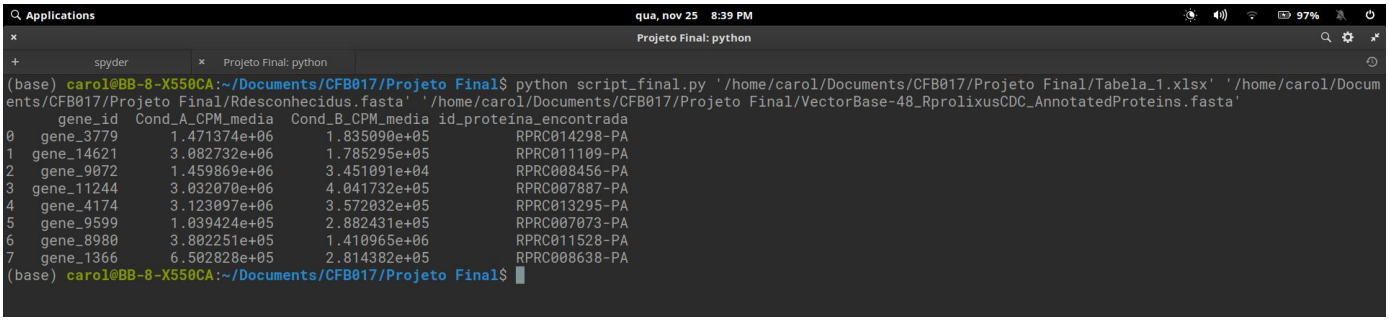
Alunas: Anna Carolina Silva Garcia e Beatriz Pereira da Silva e Souza

```
## ITEM D:
# ascending = [False] deixa a tabela ordenada em ordem decrescente.
# .head() nos dá os 5 primeiros itens.
Cond_A = RNA_seq.sort_values(['Cond_A_CPM_media'], ascending = [False]).head()
Cond_B = RNA_seq.sort_values(['Cond_B_CPM_media'], ascending = [False]).head()
# incluindo genes_id em uma lista:
A = Cond_A['gene_id'].tolist()
B = Cond_B['gene_id'].tolist()
# há genes que estão expressos no top 5 em ambas as condições.
# gene_4174 e gene_11244
# set() para retirar os genes duplicados:
genes = list(set(A+B))

## ITEM E:
record_list = []
# comparando os id da tabela com os id do multifasta desconhecido:
for line in Rhodnius_desconhecido:
    for i in genes:
        if i == line.id:
# gerando um novo arquivo fasta apenas com os genes mais expressos da tabela
(lista genes):
            record = SeqRecord(line.seq, line.id, description='')
# para gerar um multifasta pode acrescentar divesros records em uma lista:
            record_list.append(record)
# O SeqIO.write entende a lista como um multifasta e gera um novo arquivo:
SeqIO.write(record_list, 'Most_expressed_genes.fasta',"fasta")

# BLAST X

Blast_Projeto_Final =
r"/home/carol/Documents/CFB017/ProjetoFinal/Blast_Projeto_Final.txt"
meu_blast =
NcbiblastxCommandline(cmd = blastx ,query =
'/home/carol/Documents/CFB017/Projeto Final/Most_expressed_genes.fasta' ,
subject = Rhodnius_data , evalue = 0.05, outfmt = 6, out =
Blast_Projeto_Final)
# redirecionando resultados:
stdout, stdeer = meu_blast()
# adicionando cabeçalho à tabela de resultados do blastx:
blast = pd.read_csv("/home/carol/Documents/CFB017/Projeto
Final/Blast_Projeto_Final.txt", sep='\t',
names=["qseqid","sseqid","pident","length","mismatch","gapopen","qstart","qen
d","sstart","send","evalue","bitscore"])

## ITEM F:
# a tabela do blast já vem em ordem crescente de gene_id.
result = blast[['qseqid','sseqid','evalue','bitscore']]
```

```
# checando se há valor duplicado na coluna de bitscore:
# if: sem condição: booleano (true ou false)
if result['bitscore'].duplicated().any():
# ordem descrescente de e-value; retira as duplicatas baseado no bitscore
  df = result.sort_values('evalue', ascending = False).drop_duplicates(subset=['qseqid'], keep='first')
else:
# ordem descrescente de bitscore, retirada total de duplcatas pelo id do gene.
#  checando a tabela, não houve resultado igual de bitscore (os gentes duplicados foram retirados antes - linha 56).
  df = result.sort_values('bitscore', ascending = False).drop_duplicates('qseqid')
# renomeando para realizar o merge().
df = df.rename(columns={'qseqid': 'gene_id', 'sseqid': 'id_proteína_encontrada'})


## ITEM G:
#  selecionando as colunas relacionadas aos genes mais expressos (tabela original):
df1 = pd.DataFrame({'gene_id':genes}).merge(RNA_seq[['gene_id','Cond_A_CPM_media','Cond_B_CPM_media']])
# realizando merge com os reultados do BLAST e a seleção do maior hit p/ cada gene:
tabela_final = df1.merge(df[['gene_id','id_proteína_encontrada']])
# imprimindo a tabela final:
print (tabela_final)
```

Alunas: Anna Carolina Silva Garcia e Beatriz Pereira da Silva e Souza

Output (Terminal):



Output(IDE):