SCS4215

Computational Biology

Information retrieval and sequence analysis

Q1 : COX-2 (prostaglandin H2 synthase-2 (PTGS2)) gene

COX-2 has been thoroughly studied because of its role in prostaglandin synthesis. Prostaglandins have a wide range of roles in our body from aiding in digestion to propagating pain and inflammation.

Aspirin is a general inhibitor of prostaglandin synthesis and therefore, helps reduce pain.

However, aspirin also inhibits the synthesis of prostaglandins that aid in digestion. Therefore, aspirin is a poor choice for pain and inflammation management for those with ulcers or other digestion problems.

Recent advances in targeting specific prostaglandin-synthesizing enzymes have lead to the development of Celebrex, which is marketed as an arthritis therapy. Celebrex is a potent and specific inhibitor of COX-2. Celebrex is considered specific because it doesn't inhibit COX-1, which is involved in synthesizing prostaglandins that aid in digestion.

This is a remarkable accomplishment given the great similarity between COX-1 and COX-2.

This achievement has paved the way for developing new therapies that bind more specifically to their target and therefore have fewer side effects. Understanding the enzyme structures of COX-1 and COX-2 helped researchers develop a drug that would only bind and inhibit COX-2. Many of the types of information and tools used by researchers for these types of studies are freely available on the web .

GenBank, SwissProt, Sequence Manipulation suite are some of the websites.

i. **Access the entries for Human PTGS1 and PTGS2 in the "Gene" database at the NCBI (https://www.ncbi.nlm.nih.gov/) Website.**
   a. **PTGS1 and PTGS2 are isozymes. Isozymes catalyze the same reaction but are separate genes. What types of reactions do PTGS enzymes catalyze? Also, what pathway are these enzymes a part of?**
      - Catalyze the conversion of arachidonic acid to prostaglandin H2
      - Prostaglandin synthesis pathway

   b. **How is the expression of PTGS1 and PTGS2 different?**
      - Biased expression in skin (RPKM 36.8), esophagus (RPKM 21.4) and 12 other tissues
      - Biased expression in bone marrow (RPKM 59.3), urinary bladder (RPKM 41.1) and 11 other tissues
   c. **Which isozyme (PTGS1 or PTGS2) is required to inhibit inflammation?**
      - PTGS2
   d. **The drug Celebrex selectively inhibits PTGS2 while aspirin and other NSAID's inhibit both PTGS1 and PTGS2 in the same way. Why do you think researchers wanted to discover a selective inhibitor to PTGS2?**

- to discover a selective inhibitor for PTGS2 to avoid inhibiting PTGS1, which is involved in synthesizing prostaglandins for digestion
- Reduces side effects

e. Describe how studying 3-D structures of PTGS1 and PTGS2 could help researchers design a drug that binds to PTGS1, but not to PTGS2.

- Understanding structural nuances, such as flexibility and molecular interactions, enables the development of drugs with high specificity, reducing the risk of off-target effects

ii. Considering the Homo sapiens PTGS2 gene entry in NCBI gene https://www.ncbi.nlm.nih.gov/gene/ database,

a. What is the gene name?

- PTGS2

b. What is the GeneID number?

- 5743

c. Where in the human genome is this gene located?

- chromosome 1

d. What is the RefSeq accession number for the mRNA sequence of H o m o sa pie n s prostaglandin-endoperoxide synthase 2?

- NM_000963.3

e. Download the prostaglandin-endoperoxide synthase 2 Reference mRNA sequence in "FASTA" format.

gene_id: 5743
gene_symbol: PTGS2
description: prostaglandin-endoperoxide synthase 2
scientific_name: Homo sapiens
common_name: human
tax_id: 9606
genomic_range: NC_000001.11:186671791-186680423;NC_060925.1:186026616-186035248
orientation: -;-
location: chr 1
gene_type: PROTEIN_CODING
transcript_accession: NM_000963.4
transcript_name:
transcript_length: 4510
transcript_cds_coords: NM_000963.4:134-1948
protein_accession: NP_000954.1
isoform_name:
protein_length: 604
protein_name: prostaglandin G/H synthase 2 precursor

f. What is the RefSeq accession number for the H o m o sa pie n s PTGS2 protein sequence? Download the sequence in "FASTA" format.

- NP_000954.1

iii. **Search for the UniProt entry for PTGS2 in Expasy https://www.expasy.org/website.**
   a. **What are the alternate names for this protein.**
      - Prostaglandin-endoperoxide synthase 2, Cyclooxygenase-2
   b. **What types of drugs target this protein?**
      - Nonsteroidal anti-inflammatory drugs
   c. **What amino acid is acetylated by aspirin (amino acid type)?**
      - Serine

iv. Translate the mRNA sequence of PTGS2 into Protein. Use "Translate " tool in ExPASy. Explain the output.

Readings:

http://www.aspree.org/AUS/aspree-content/aspirin/how-aspirin-works.aspx

Q2. Python Exercises

1. **Below shows some files with embedded sample names:**

lane1_NewCode_L001_R1.fastq.gz

lane1_NoIndex_L001_R1.fastq.gz

lane1_NoIndex_L001_R2.fastq.gz

pipeline_processing_output.log

lane7027_ACTGAT_JH25_L001_R1.fastq.gz

lane7027_ACTTGA_E30_1_2_Hap4_24h_L001_R1.fastq.gz

lane7027_AGTTCC_JH14_L001_R1.fastq.gz

lane7027_CGGAAT_JH37_L001_R1.fastq.gz

lane7027_GCCAAT_**E30_1_2l_Hap4_log**_L001_R1.fastq.gz

lane7127_GGCTAC_E30_1_4_Hap4_48h_L001_R1.fastq.gz

Write a Python code to extract the sample name from these files ignoring any files which do not match the format given below.

The format is:

1. **Written lane number**
2. **Barcode**
3. **Sample name**

4. **Numeric lane number (starting with L)**
5. **Read number (R1/2/3/4)**
6. **File extension**

Eg. Lane8127_GCCAAT_S30_1_2l_Hap4_log_L001_R1.fastq.gz the sample name would be, S30_1_2l_Hap4_log

```
import re


def extract_sample_name(file_name):
    # Define the regular expression pattern
    pattern = r'^lane\d+_([A-Z\d]+)_([A-Za-z\d_]+)_L\d+_([R]\d+)\.fastq\.gz$'

    # Try to match the pattern in the file name
    match = re.match(pattern, file_name)

    # If there is a match, extract and return the sample name
    if match:
        return match.group(2)
    else:
        return "no"

# List of file names
files = [
    "lane1_NewCode_L001_R1.fastq.gz",
    "lane1_NoIndex_L001_R1.fastq.gz",
    "lane1_NoIndex_L001_R2.fastq.gz",
    "pipeline_processing_output.log",
    "lane7027_ACTGAT_JH25_L001_R1.fastq.gz",
    "lane7027_ACTTGA_E30_1_2_Hap4_24h_L001_R1.fastq.gz",
```

```
    "lane7027_AGTTCC_JH14_L001_R1.fastq.gz",
    "lane7027_CGGAAT_JH37_L001_R1.fastq.gz",
    "lane7027_GCCAAT_E30_1_2l_Hap4_log_L001_R1.fastq.gz",
    "lane7127_GGCTAC_E30_1_4_Hap4_48h_L001_R1.fastq.gz",
    "lane8127_GCCAAT_S30_1_2l_Hap4_log_L001_R1.fastq.gz"
]


# Extract sample names from each file
sample_names = [extract_sample_name(file_name) for file_name in files]


# Display the result
for file_name, sample_name in zip(files, sample_names):
    print(f"{file_name} -> {sample_name}")
```

Outputs

```
PS D:\Chathura\UGVLe\4Y2S\SCS4215 Computational Biology\Labsheets\1> & C:/Users/ASUS/AppData/Local/Programs/Python/P
ython310/python.exe "d:/Chathura/UGVLe/4Y2S/SCS4215 Computational Biology/Labsheets/1/sample_name_extract.py"
lane1_NewCode_L001_R1.fastq.gz -> no
lane1_NoIndex_L001_R1.fastq.gz -> no
lane1_NoIndex_L001_R2.fastq.gz -> no
pipeline_processing_output.log -> no
lane7027_ACTGAT_JH25_L001_R1.fastq.gz -> JH25
lane7027_ACTTGA_E30_1_2_Hap4_24h_L001_R1.fastq.gz -> E30_1_2_Hap4_24h
lane7027_AGTTCC_JH14_L001_R1.fastq.gz -> JH14
lane7027_CGGAAT_JH37_L001_R1.fastq.gz -> JH37
lane7027_GCCAAT_E30_1_2l_Hap4_log_L001_R1.fastq.gz -> E30_1_2l_Hap4_log
lane7127_GGCTAC_E30_1_4_Hap4_48h_L001_R1.fastq.gz -> E30_1_4_Hap4_48h
lane8127_GCCAAT_S30_1_2l_Hap4_log_L001_R1.fastq.gz -> S30_1_2l_Hap4_log
PS D:\Chathura\UGVLe\4Y2S\SCS4215 Computational Biology\Labsheets\1>
```

2. Create a FASTA file by obtaining 10 Dengue 1- Envelop gene DNA sequences from NCBI. Write a Python-program that reads the FASTA file, cleans up the header line to have only Accession number & gene-name and print headers and sequences to standard output as multi-FASTA-file again.

3. Write a Python program to search the DNA Sequence for the presence of one of the following Transcription Factor Binding Sites(TFBS) with ambiguity codes. Search for all the positions in the sequence where TFBS is located.

Code	Represents
A	Adenine
G	Guanine
C	Cytosine
T	Thymine
Y	Pyrimidine (C or T)
R	Purine (A or G)
W	weak (A or T)
S	strong (G or C)
K	keto (T or G)
M	amino (C or A)
D	A, G, T (not C)
V	A, C, G (not T)
H	A, C, T (not G)
B	C, G, T (not A)

Transcription Factor	**Consensus Sequence**
RUNX1	BHTGTGGTYW
TGIF1	WGACAGB
IKZF1	BTGGGARD

The sequence is shown below.

>search_seq

GACACCTCAGTACTAGGATGNNNNNNTATCAGCCTGAACTAGCAGGCCTGGTTCCAAATT
TTTTTATCAACACTCGTAGGGGGATTATCCTAGAGGGGGTCTGGGATTTCTTTGACATCA
GAGTATTTTTGCCTTGCTCCTTCACAATTTGGGAACAAATAATTTAGTGGTTATTAACCC
TGGCTACGCACTGGAAACTTTAAAAATAATGCTGGTATGAAATTTACACAGAGTATCGTG
AAAATTTTCACTGAGTACCATGTGGTTATACATTGGATAAGGCTCCAGGAAGCAGCTACT
GGAAGACAGCCATGCCAAGAGTGGTTAGTGGTTGGAATTTTGGCAAGTCAGTTTTAGTCT
GCCTTATCAAATACATGGGCATACAGATAAATCCTTAGATGGCTCTCCTACTTACTGAAA
CATTTTCTATCTATCTATCTATCTATCTATCTATTTGGGAAGCTATCTATCTATCTATCA
TTTATTTAAGGTAGTCTCTATCTGCCTCTGTCTCTGTCTGTCTCTGTGTCTCTGTGTCTG
TCTGCTCTCTCTCTCTCTCTGTGGGAATCTCTCTCTGTGTGTGTGTGTGTATGTGTGTGT
GTGTGTGTGTGGTGTGCATGAACATGAGTAAAATCCATAAGGAAACTTTCAGAGTTGGTC
CTCTCCTTATATCAAATGGATCCAGGAATTAAACTCAGGTTCAATTCTTGGTGCCTTTAC
TAGTTGAGCCATCTCACTGGCTCTTCATCATCTTTAGAATAAACTCACTTTATTACACAC
ACACACACACACACAACCTGGGAGTACACACACACACACAACCAAAGCCCCAACGGAAAA
CTACAATATTATAATGAATACACAGGTTCTCAACATAGTCTCTGCCACGCTTGCAGACAA
AGATGAGTAGAAGTAGAAAGAACCAGGGAAACGTGGAGCAAGTCAGAAGGAATAACAGTC
AGAAGGAATAACAGTCAGAAGGAATAACAGTCAGAAGGAGTAACAGTCAGAAGGAATAGC
AGTCAGAAGGAATAACAGTCAGAAGACAGCACAGTCAGAAGGAATAACAGTCAGAAGGAA
TAACAGTCAGAAGGAATAACAGTCAGAAGGAATAACAGTCAGAAGGAATAGCAGTCAGAA
GGAATAACAGTCAGAAGGAATAACAGTCAGAAGGAATAACAGTCAAAGAAATAGCAGTCA

GAAGGAATAGCAGTCAGAAGGAATAACAGTCAAAGGAGCAGTCAGAAGGAGTAACAGTCA
GAAGGAATAACAGTCAGAAGGAATAACAGTCAAAGGAATAGCAGTCAGAAGGAGTAACAG
TCAGAGCAAACACAGAGATGACAAAGGCAATGGGGTCAGAGACTTCACCACTCTCCAAGA

```
      import re


def search_tfbs(sequence, tfbs_dict):
    # Remove newlines and spaces from the sequence
    sequence = "".join(sequence.split())

    positions = {}
    for tf, consensus_sequence in tfbs_dict.items():
        # Convert ambiguity codes to regular expressions
        consensus_sequence = consensus_sequence.replace('B', '[CGT]')
        consensus_sequence = consensus_sequence.replace('D', '[AGT]')
        consensus_sequence = consensus_sequence.replace('H', '[ACT]')
        consensus_sequence = consensus_sequence.replace('K', '[GT]')
        consensus_sequence = consensus_sequence.replace('M', '[AC]')
        consensus_sequence = consensus_sequence.replace('N', '[ACGT]')
        consensus_sequence = consensus_sequence.replace('R', '[AG]')
        consensus_sequence = consensus_sequence.replace('S', '[CG]')
        consensus_sequence = consensus_sequence.replace('V', '[ACG]')
        consensus_sequence = consensus_sequence.replace('W', '[AT]')
        consensus_sequence = consensus_sequence.replace('Y', '[CT]')

        matches = [match.start() for match in
re.finditer(f'(?={consensus_sequence})', sequence)]
        if matches:
            positions[tf] = matches
    return positions

if __name__ == "__main__":
    search_seq = """
    GACACCTCAGTACTAGGATGNNNNNNTATCAGCCTGAACTAGCAGGCCTGGTTCCAAATT
    TTTTTATCAACACTCGTAGGGGGATTATCCTAGAGGGGGTCTGGGATTTCTTTGACATCA
    GAGTATTTTTGCCTTGCTCCTTCACAATTTGGGAACAAATAATTTAGTGGTTATTAACCC
    TGGCTACGCACTGGAAACTTTAAAAATAATGCTGGTATGAAATTTACACAGAGTATCGTG
    AAAATTTTCACTGAGTACCATGTGGTTATACATTGGATAAGGCTCCAGGAAGCAGCTACT
    GGAAGACAGCCATGCCAAGAGTGGTTAGTGGTTGGAATTTTGGCAAGTCAGTTTTAGTCT
    GCCTTATCAAATACATGGGCATACAGATAAATCCTTAGATGGCTCTCCTACTTACTGAAA
    CATTTTCTATCTATCTATCTATCTATCTATCTATTTGGGAAGCTATCTATCTATCTATCA
    TTTATTTAAGGTAGTCTCTATCTGCCTCTGTCTCTGTCTGTCTCTGTGTCTCTGTGTCTG
    TCTGCTCTCTCTCTCTCTCTGTGGGAATCTCTCTCTGTGTGTGTGTGTGTATGTGTGTGT
    GTGTGTGTGTGGTGTGCATGAACATGAGTAAAATCCATAAGGAAACTTTCAGAGTTGGTC
```

```
CCTCTCCTTATATCAAATGGATCCAGGAATTAAACTCAGGTTCAATTCTTGGTGCCTTTAC
TAGTTGAGCCATCTCACTGGCTCTTCATCATCTTTAGAATAAACTCACTTTATTACACAC
ACACACACACACACAACCTGGGAGTACACACACACACACAACCAAAGCCCCAACGGAAAA
CTACAATATTATAATGAATACACAGGTTCTCAACATAGTCTCTGCCACGCTTGCAGACAA
AGATGAGTAGAAGTAGAAAGAACCAGGGAAACGTGGAGCAAGTCAGAAGGAATAACAGTC
AGAAGGAATAACAGTCAGAAGGAATAACAGTCAGAAGGAGTAACAGTCAGAAGGAATAGC
AGTCAGAAGGAATAACAGTCAGAAGACAGCACAGTCAGAAGGAATAACAGTCAGAAGGAA
TAACAGTCAGAAGGAATAACAGTCAGAAGGAATAACAGTCAGAAGGAATAGCAGTCAGAA
GGAATAACAGTCAGAAGGAATAACAGTCAGAAGGAATAACAGTCAAAGAAATAGCAGTCA
GAAGGAATAGCAGTCAGAAGGAATAACAGTCAAAGGAGCAGTCAGAAGGAGTAACAGTCA
GAAGGAATAACAGTCAGAAGGAATAACAGTCAAAGGAATAGCAGTCAGAAGGAGTAACAG
TCAGAGCAAACACAGAGATGACAAAGGCAATGGGGTCAGAGACTTCACCACTCTCCAAGA
"""

tfbs_dict = {
    "RUNX1": "BHTGTGGTYW",
    "TGIF1": "WGACAGB",
    "IKZF1": "BTGGGARD"
}

positions = search_tfbs(search_seq, tfbs_dict)

for tf, tf_positions in positions.items():
    print(f"{tf} found at positions: {tf_positions}")
```

Outputs

```
PS D:\Chathura\UGVLe\4Y2S\SCS4215 Computational Biology\Labsheets\1> & C:/Users/ASUS/AppData/Local/Programs/Python/Python310/python.exe "d:/Chathura/UGVLe/4Y2S/SCS4215 Computational Biology/Labsheets/1/TFBS.py"
RUNX1 found at positions: [258]
TGIF1 found at positions: [303, 1044]
IKZF1 found at positions: [454, 560, 798]
PS D:\Chathura\UGVLe\4Y2S\SCS4215 Computational Biology\Labsheets\1>
```

Q3 – Biopython

Biopython Tutorial and Cookbook https://biopython.org/DIST/docs/tutorial/Tutorial.html#sec2

1. Write a Biopython program that asks the user to input a DNA-sequence and then translates the sequence to protein sequence.

```
from Bio.Seq import Seq

while True:
dna_sequence = input("Enter DNA sequence (or type 'exit' to end): ")
```

```
if dna_sequence.lower() == 'exit':
print("Exiting the program.")
break

try:
# Translate to protein sequence
protein_sequence = Seq(dna_sequence).translate()

# Print the protein sequence
print(f"Translated Protein Sequence: {protein_sequence}")
except Exception as e:
print(f"Error: {e}")
```

Outputs

```
Enter DNA sequence (or type 'exit' to end): AUGGGAUGUCGCCGAAAC
Translated Protein Sequence: MGCRRN
Enter DNA sequence (or type 'exit' to end):
```

Write a Biopython program that will find all articles related to Alzheimer's in PubMed. Print the total number of articles available and the authors.

```
from Bio import Entrez

def search_pubmed(query, max_results=10):
Entrez.email = "chathura.manoharas@gmail.com"
handle = Entrez.esearch(db="pubmed", term=query, retmax=max_results)
record = Entrez.read(handle)
handle.close()
return record

def fetch_pubmed_details(id_list):
ids = ",".join(id_list)
handle = Entrez.efetch(db="pubmed", id=ids, rettype="medline",
retmode="text")
records = handle.read()
handle.close()
return records

def parse_pubmed_records(records):
authors_list = []
for record in records.split("\n\nPMID")[1:]:
authors = []
for line in record.split('\n'):
if line.startswith('AU  - '):
authors.append(line[6:])
authors_list.append(authors)
return authors_list

if __name__ == "__main__":
query = "Alzheimer's"

# Search PubMed
search_results = search_pubmed(query)
# Print the total articles
total_articles = int(search_results["Count"])
print(f"Total number of articles related to Alzheimer's: {total_articles}")


id_list = search_results["IdList"][:10]
pubmed_records = fetch_pubmed_details(id_list)
# Parse and print authors
authors_list = parse_pubmed_records(pubmed_records)
# Print authors
for i, authors in enumerate(authors_list, 1):
```

```
print(f"\nAuthors for Article {i}:")
for author in authors:
print(f" - {author}")
2.
```

Outputs

```
PS D:\Chathura\UGVLe\4Y2S\SCS4215 Computational Biology\Labsheets\1> & C:/Users/ASUS/AppData/Local/Programs/Python/Python310/python.exe "d:/Chathura/UGVLe/4Y2S/SCS4215 Computational Biology/Labsheets/1/findArticles.py"
Total number of articles related to Alzheimer's: 223320

Authors for Article 1:
 - Kusama T
 - Takeuchi K
 - Kiuchi S
 - Aida J
 - Osaka K

Authors for Article 2:
 - Weijs RW
 - Oudegeest-Sander MH
 - Hopman MT
 - Thijssen DH
 - Claassen JA

Authors for Article 3:
 - Shateri S
 - Khatami SH
 - HaghbinToutounchi A
 - Rajaei S
 - Mahdavi M
 - MahmoodiBaram S
 - Shahidi GA
 - Habibi AH
 - Aghamollaii V
 - Ghlichnia B
 - Safakish L
 - Doagoo A
 - Salmani F
 - Tafakhori A
 - Keramatinia A
 - Shahmohammadi MR
 - Karima S

Authors for Article 4:
 - Bhattacharyya R
 - Jha BK

Authors for Article 5:
 - Galgani A
```

3. Write a Biopython-program that finds CpG-islands from a given DNA-sequence.

```
from Bio.SeqUtils import gc_fraction as GC

def find_cpg_islands(dna_sequence, threshold=50, window_size=200,
step_size=10):
    cpg_islands = []

    for start in range(0, len(dna_sequence) - window_size + 1, step_size):
        end = start + window_size
        gc_content = GC(dna_sequence[start:end])

        if gc_content > threshold:
            cpg_islands.append((start, end))

    return cpg_islands

dna_sequence = input("Enter DNA sequence: ")

# Find CpG islands
cpg_islands = find_cpg_islands(dna_sequence)

# Print CpG islands
print("CpG Islands:")
for island in cpg_islands:
    print(f"Start: {island[0]}, End: {island[1]}")
4.
```