ACCESS BIOINFORMATICS DATABASES WITH BIO-PYTHON

This project is aimed to deploy python-based programming pipelines and scripts to automate biological data retrieval and analysis.

1. NCBI

This section uses the NCBI module from Biopyton to perform BLAST (Basic Local Alignment Search Tool). This finds regions of local similarity between sequences from the NCBI databse. It is a preliminary analysis in identifying the strain, species, or the source, to which our isolated sequence belongs to or is closely related.

Import Modules

In [19]:
```python
from Bio.Blast import NCBIWWW
from Bio import SeqIO, SearchIO
```

1.1. Nucleotide BLAST

First, I performed nucleotide BLAST using the nucleotide sequence stored in the nuc_seq.fasta file.
As shown, the sequence is 774 base pairs long:

In [20]:
```python
nuc_record = SeqIO.read("nuc_seq.fasta", format = "fasta")
len(nuc_record)
```

Out[20]:
```
774
```

We can access various parts of the sequences file. The file is composed of two lines: the description and the sequence.

1. The description:

In [21]:
```python
nuc_record.description
```

Out[21]:
```
'MT598137.1 Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/IRN/PN-2142-S/2020 surface glyco
protein (S) gene, partial cds'
```

Our sequence is a gene encoding surface glycoprotein SARS-CoV-2 isolate.

2. The sequence:

In [22]:
```python
nuc_record.seq
```

Out[22]:
```
Seq('ATCGCTCCAGGGCAAACTGGAAAGATTGCTGATTATAATTATAAATTACCAGAT...GGT')
```

Now we perform blast!

The module takes in three parameters:
1. 'blastn'which stands for BLAST nucleotide.
2. The database used it 'nt' which stands for nucleotide.
3. 'nuc_record.seq' which is our record.

The SEarchIO Biopython module is used to read in the results of our BLAST.

In [23]:
```python
result_handle = NCBIWWW.qblast("blastn","nt",nuc_record.seq)
blast_result = SearchIO.read(result_handle,"blast-xml")
```

```
C:\Users\fxy40\anaconda3\lib\site-packages\Bio\SearchIO\_legacy\__init__.py:12: BiopythonDeprecationWarning: The 'Bi
o.SearchIO._legacy' module for parsing BLAST plain text output is deprecated and will be removed in a future release
of Biopython. Consider generating your BLAST output for parsing as XML or tabular format instead.
  warnings.warn(
```

I used the Python indexing technique to print out the first two results:

In [24]:
```python
print(blast_result[0:2])
```

```
Program: blastn (2.14.1+)
  Query: No (774)
         definition line
 Target: nt
   Hits: ----  -----  ----------------------------------------------------------
            #  # HSP  ID + description
         ----  -----  ----------------------------------------------------------
            0      1  gi|2529195153|gb|OR223350.1|  Severe acute respiratory ...
            1      1  gi|2529195140|gb|OR223349.1|  Severe acute respiratory ...
```

Usually, the first hit the the BLAST constitutes the best match. Hence, we will fetch more details about the first sequence from the BLAST results.

In [25]:
```python
Seq = blast_result[0]
print(f"Sequence ID: {Seq.id}")
print(f"Sequence Description: {Seq.description}")

details = Seq[0]
print(f"E-value: {details.evalue}")
```

```
Sequence ID: gi|2529195153|gb|OR223350.1|
Sequence Description: Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/USA/WA-UW143/2020 ORF1
ab polyprotein (ORF1ab), ORF1a polyprotein (ORF1ab), surface glycoprotein (S), ORF3a protein (ORF3a), envelope protei
n (E), membrane glycoprotein (M), ORF6 protein (ORF6), ORF7a protein (ORF7a), ORF7b (ORF7b), ORF8 protein (ORF8), nuc
leocapsid phosphoprotein (N), and ORF10 protein (ORF10) genes, complete cds
E-value: 0.0
```

A low value for a sequence of considerable length is considered as optimal. Here, the E-valye is 0.0. Therefore, the sequence is an exct or a very closely related match to our sequence.

In [26]:
```python
print(f"alignment:\n{details.aln}")
```

```
alignment:
Alignment with 2 rows and 774 columns
ATCGCTCCAGGGCAAACTGGAAAGATTGCTGATTATAATTATAA...GGT No
ATCGCTCCAGGGCAAACTGGAAAGATTGCTGATTATAATTATAA...GGT gi|2529195153|gb|OR223350.1|
```

This gives us the alignment of the two sequences, with the top one being the query sequence which we used, and the bottom the result which we got.

1.2. Protein BLAST

In [27]:
```python
prot_record = SeqIO.read("prot_seq.fasta", format="fasta")
len(prot_record)
```

Out[27]:
```
258
```

Length of the protein sequence: 258 amino acids.

This module also takes three parameters:
1. 'blastp' stands for BLAST protein.
2. The database used it PDB.
3. 'prot_record.seq' is our protein record.

In [28]:
```python
result_handle = NCBIWWW.qblast("blastp", "pdb", prot_record.seq)
blast_result = SearchIO.read(result_handle, "blast-xml")
```

The first two results are fetched by running the next two cells:

In [29]:
```
print(blast_result[0:2])
```

```
Program: blastp (2.14.1+)
  Query: unnamed (258)
         protein product
 Target: pdb
   Hits: ----  -----  ----------------------------------------------------------
            #  # HSP  ID + description
         ----  -----  ----------------------------------------------------------
            0      1  pdb|7CAB|A  Chain A, Spike glycoprotein [Severe acute r...
            1      1  pdb|7R4I|A  Chain A, Spike glycoprotein [Severe acute r...
```

Our protein sequence corresponds to two matches which have the greatest similarity, among which we fetch the first result:

In [30]:
```
Seq = blast_result [0]
print(f"Sequence ID: {Seq.id}")
print(f"Sequence Description: {Seq.description}")

details = Seq[0]
print(f"E-value: {details.evalue}")
```

```
Sequence ID: pdb|7CAB|A
Sequence Description: Chain A, Spike glycoprotein [Severe acute respiratory syndrome coronavirus 2]
E-value: 0.0
```

To check for alignment, I ran this cell which returned the alignment of our sequence against the results fetched using BLAST.

In [31]:
```
print(f"alignment:\n {details.aln}")
```

```
alignment:
 Alignment with 2 rows and 258 columns
IAPGQTGKIADYNYKLPDDFTGCVIAWNSNNLDSKVGGNYNYLY...PIG unnamed
IAPGQTGKIADYNYKLPDDFTGCVIAWNSNNLDSKVGGNYNYLY...PIG pdb|7CAB|A
```

2. ENTREZ

This section fetches PUBMED literature and nucleotide sequences using ENTREZ. ENTREZ is NCBI's primary text search and retrieval system that integrates the PUBMED database of biomedical literature wtih 38 other literature and molecular databases, including DNA and protein sequences, structure, gene, genome, genetic variation and gene expression.

Import Modules

In [32]:
```
from Bio import Entrez
```

Below gives the details about the kind of parameters the Entrez module can use, and the kind of data we can fetch using this module.

In [33]:
```
help(Entrez)
```

```
Help on package Bio.Entrez in Bio:

NAME
    Bio.Entrez - Provides code to access NCBI over the WWW.

DESCRIPTION
    The main Entrez web page is available at:
    http://www.ncbi.nlm.nih.gov/Entrez/ (http://www.ncbi.nlm.nih.gov/Entrez/)

    Entrez Programming Utilities web page is available at:
    http://www.ncbi.nlm.nih.gov/books/NBK25501/ (http://www.ncbi.nlm.nih.gov/books/NBK25501/)

    This module provides a number of functions like ``efetch`` (short for
    Entrez Fetch) which will return the data as a handle object. This is
    a standard interface used in Python for reading data from a file, or
    in this case a remote network connection, and provides methods like
    ``.read()`` or offers iteration over the contents line by line. See
    also "What the heck is a handle?" in the Biopython Tutorial and
    Cookbook: http://biopython.org/DIST/docs/tutorial/Tutorial.html (http://biopython.org/DIST/docs/tutorial/Tutor
```

Entrez requires users to put in their emails to run queries:

In [34]:
```
Entrez.email = "jfxy0518@gmail.com"
```

I ran the code below to identify the types of databases which can be accessed using the Entrez module.

In [35]:
```python
handle = Entrez.einfo()
record = Entrez.read(handle)
record["DbList"]
```

Out[35]:
```
['pubmed', 'protein', 'nuccore', 'ipg', 'nucleotide', 'structure', 'genome', 'annotinfo', 'assembly', 'bioproject',
'biosample', 'blastdbinfo', 'books', 'cdd', 'clinvar', 'gap', 'gapplus', 'grasp', 'dbvar', 'gene', 'gds', 'geoprofile
s', 'homologene', 'medgen', 'mesh', 'nlmcatalog', 'omim', 'orgtrack', 'pmc', 'popset', 'proteinclusters', 'pcassay',
'protfam', 'pccompound', 'pcsubstance', 'seqannot', 'snp', 'sra', 'taxonomy', 'biocollections', 'gtr']
```

2.1. PUBMED

In this step, I attempt to fetch PubMed literature data.

In [36]:
```python
handle = Entrez.einfo(db="pubmed")
record = Entrez.read(handle)
record["DbInfo"]["Description"]
```

Out[36]:
```
'PubMed bibliographic record'
```

As shown, PubMed is a bibliographic database containing bibliographic records...

In [37]:
```python
record["DbInfo"]["Count"]
```

Out[37]:
```
'35953109'
```

...with a count of 35937484 bibliographic records in this database (as of July 13th, 2023).

I then filtered the data of choice by using the e-search module of Entrez. This fetches all the literature or data containing the term 'biopython' in their title:

In [38]:
```python
handle = Entrez.esearch(db="pubmed", term="biopython")
record = Entrez.read(handle)
record["IdList"]
```

Out[38]:
```
['36818783', '36245797', '36094101', '35497637', '35496474', '35402671', '34735950', '34484417', '34434786', '3418901
2', '33994075', '33902722', '33809815', '33242467', '32044951', '31762715', '31278684', '31069053', '30013827', '2964
1230']
```

An alternative way to fetch literature details is to use the IdList instead of the term 'biopython'. To do that, I replaced the parameter 'esearch' with 'esummary', and pass in the IDs I got from the previous search. I formatted it into a specific type, showing the author, title, publication date and journal name using the *for* loop:

In [39]:
```python
handle = Entrez.esummary(db="pubmed", id='36818783,36245797')
records = Entrez.parse(handle)


for record in records:
    print(record['AuthorList'],record['Title'],record['PubDate'],record['FullJournalName'])
```

```
['Olds CG', 'Berta-Thompson JW', 'Loucks JJ', 'Levy RA', 'Wilson AW'] Applying a modified metabarcoding approach for
the sequencing of macrofungal specimens from fungarium collections. 2023 Jan-Feb Applications in plant sciences
['Nallasamy V', 'Seshiah M'] Energy Profile Bayes and Thompson Optimized Convolutional Neural Network protein structu
re prediction. 2023 Neural computing & applications
```

2.2. Nucleotide

In this section, I fetch nucleotide sequence records using the same Entrez module. Within the e-search parameter, I put 'nucleotide' as my database, and set it to retrieve 10 records. The ID list of all the nucleotide sequences which have the term 'severe acute respiratory syndrome' is found:

In [40]:
```python
handle = Entrez.esearch(db="nucleotide",retmax=10, term="Severe acute respiratory syndrome")
record = Entrez.read(handle)
record["IdList"]
```

Out[40]:
```
['2542475402', '2542475384', '2542404305', '2542404289', '2542404272', '2542404258', '2542404242', '2542404228', '254
2404213', '2542404198']
```

Then, I fetched the record of these individual IDs. The retrieval type is set to 'gb' (Genbank), and the output is set to be in a text format:

```
In [41]: handle = Entrez.efetch(db="nucleotide", id="2531721439", rettype="gb", retmode="text")
         print(handle.read())
```

```
LOCUS       LC773238               29685 bp    RNA     linear   VRL 08-JUL-2023
DEFINITION  Severe acute respiratory syndrome coronavirus 2
            SARS-CoV-2/human/Japan/kmumc011145/2023 RNA, nearly complete
            genome.
ACCESSION   LC773238
VERSION     LC773238.1
DBLINK      BioProject: PRJDB16147
            BioSample: SAMD00627887
            Sequence Read Archive: DRR489579
KEYWORDS    .
SOURCE      Severe acute respiratory syndrome coronavirus 2
  ORGANISM  Severe acute respiratory syndrome coronavirus 2
            Viruses; Riboviria; Orthornavirae; Pisuviricota; Pisoniviricetes;
            Nidovirales; Cornidovirineae; Coronaviridae; Orthocoronavirinae;
            Betacoronavirus; Sarbecovirus; Severe acute respiratory
            syndrome-related coronavirus.
REFERENCE   1
  AUTHORS   Nakamori,Y. and Kashihara,M.
  TITLE     Clinical Experience of Treatment of Immunocompromised Individuals
```

Similarly, apart from the term, we can use the chaining method using regular expression. This lists the IDs associated with accD (the gene name) and the organism E. Coli:

```
In [42]: handle = Entrez.esearch(db='nucleotide', term='accD[Gene Name] AND "E. coli"[Organism]', retmax="20")
         result_list = Entrez.read(handle)
```

```
In [43]: id_list = result_list['IdList']
         count = result_list['Count']

         print(id_list)
         print("\n")
         print(count)
```

```
['2540286096', '2540285939', '2540285880', '2540285612', '2540285515', '2540285302', '2540285285', '2540281271', '254
0281265', '2540281264', '2536864279', '2535150858', '2535150857', '2535150855', '2535150854', '2535150853', '25351508
52', '2535150851', '2535150850', '2535150844']


220644
```

```
In [44]: handle.close()
```

3. PDB

This section fetches protein structures from PDB (Protein Data Bank), using the PDB module from Biopython to fetch, parse, and filter details of protein sequences from the PDB database.

Import Modules

```
In [45]: from Bio.PDB import PDBParser,PDBList
```

Details about the PDBParser module:

In [46]:
```python
help(PDBParser)
```

```
Help on class PDBParser in module Bio.PDB.PDBParser:

class PDBParser(builtins.object)
 |  PDBParser(PERMISSIVE=True, get_header=False, structure_builder=None, QUIET=False, is_pqr=False)
 |
 |  Parse a PDB file and return a Structure object.
 |
 |  Methods defined here:
 |
 |  __init__(self, PERMISSIVE=True, get_header=False, structure_builder=None, QUIET=False, is_pqr=False)
 |      Create a PDBParser object.
 |
 |      The PDB parser call a number of standard methods in an aggregated
 |      StructureBuilder object. Normally this object is instantiated by the
 |      PDBParser object itself, but if the user provides his/her own
 |      StructureBuilder object, the latter is used instead.
 |
 |      Arguments:
 |       - PERMISSIVE - Evaluated as a Boolean. If false, exceptions in
 |         constructing the SMCRA data structure are fatal. If true (DEFAULT),
 |         the exceptions are caught, but some residues or atoms will be missing.
 |         THESE EXCEPTIONS ARE DUE TO PROBLEMS IN THE PDB FILE!.
 |       - get_header - unused argument kept for historical compatibility.
 |       - structure_builder - an optional user implemented StructureBuilder class.
 |       - QUIET - Evaluated as a Boolean. If true, warnings issued in constructing
 |         the SMCRA data will be suppressed. If false (DEFAULT), they will be shown.
 |         These warnings might be indicative of problems in the PDB file!
 |       - is_pqr - Evaluated as a Boolean. Specifies the type of file to be parsed.
 |         If false (DEFAULT) a .pdb file format is assumed. Set it to true if you
 |         want to parse a .pqr file instead.
 |
 |  get_header(self)
 |      Return the header.
 |
 |  get_structure(self, id, file)
 |      Return the structure.
 |
 |      Arguments:
 |       - id - string, the id that will be used for the structure
 |       - file - name of the PDB file OR an open filehandle
 |
 |  get_trailer(self)
 |      Return the trailer.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

Details about the PDBList module:

In [47]:
```python
help(PDBList)
```

```
Help on class PDBList in module Bio.PDB.PDBList:

class PDBList(builtins.object)
 |  PDBList(server='ftp://ftp.wwpdb.org', pdb=None, obsolete_pdb=None, verbose=True)
 |
 |  Quick access to the structure lists on the PDB or its mirrors.
 |
 |  This class provides quick access to the structure lists on the
 |  PDB server or its mirrors. The structure lists contain
 |  four-letter PDB codes, indicating that structures are
 |  new, have been modified or are obsolete. The lists are released
 |  on a weekly basis.
 |
 |  It also provides a function to retrieve PDB files from the server.
 |  To use it properly, prepare a directory /pdb or the like,
 |  where PDB files are stored.
 |
 |  All available file formats (PDB, PDBx/mmCif, PDBML, mmtf) are supported.
 |  Please note that large structures (containing >62 chains
```

The structure of the 7BYR protein is downlaoded and stored:

In [48]:
```
pdbl=PDBList()
pdbl.retrieve_pdb_file("7BYR", file_format="pdb", pdir="dir")
```

```
Structure exists: 'dir\pdb7byr.ent'
```

Out[48]:
```
'dir\\pdb7byr.ent'
```

Then, the structure of the protein is read by PDBParser.

In [49]:
```
parser = PDBParser()
structure = parser.get_structure("7BYR","dir/pdb7byr.ent")
```

```
C:\Users\fxy40\anaconda3\lib\site-packages\Bio\PDB\StructureBuilder.py:89: PDBConstructionWarning: WARNING: Chain A i
s discontinuous at line 26237.
  warnings.warn(
C:\Users\fxy40\anaconda3\lib\site-packages\Bio\PDB\StructureBuilder.py:89: PDBConstructionWarning: WARNING: Chain B i
s discontinuous at line 26405.
  warnings.warn(
C:\Users\fxy40\anaconda3\lib\site-packages\Bio\PDB\StructureBuilder.py:89: PDBConstructionWarning: WARNING: Chain C i
s discontinuous at line 26545.
  warnings.warn(
```

I then used a *for* loop to identify the number of chains in the protein sequence:

In [50]:
```
for chain in structure[0]:
    print(f"chainid:{chain.id}")
```

```
chainid:A
chainid:B
chainid:C
chainid:H
chainid:L
chainid:D
chainid:E
chainid:F
chainid:G
chainid:I
chainid:J
```

As shown, the protein structure has a total of 11 chains named in alphabetical order.

The resolution of the protein is 3.84 angstroms:

In [51]:
```
resolution= structure.header["resolution"]
resolution
```

Out[51]:
```
3.84
```

Using a keyboard variable, I passed in the same structure variable containing the protein structure details. From here, I fetched the header, which has the keywords that are associated with the proteins.

In [52]:
```
keywords = structure.header["keywords"]
keywords
```

Out[52]:
```
'sars-cov-2, antigen, rbd, neutralizing antibody, viral protein'
```

As shown, the protein is a SARS-CoV-2 protein, and is associated with the keywords antigen, RBD, neutralizing antibody, and viral protein.

4. EXPASY

In this section, using the ExPASy module, I fetched domain details of the proteins that I retrieved and processed from the section above. ExPASy is an online bioinformatics resource operated by SIB (Swiss Institute of Bioinformatics). Prosite is a protein database, which consists of entries describing the protein families, domains and functional sites, as well as amino acid patterns and profiles in them.

4.1. PROSITE

Import Modules

In [53]:
```
from Bio import ExPASy
from Bio.ExPASy import Prosite
```

In this step, I passed in the prosite ID to the ExPASy module to show the contents:

In [54]:
```
handle = ExPASy.get_prosite_raw('PS51442')
record = Prosite.read(handle)
print(record.description)
```

```
Coronavirus main protease (M-pro) domain profile.
```

Various PDB structures possessing this domain profile can be found:

In [55]:
```
print(record.pdb_structs[:10])
```

```
['1LVO', '1P9S', '1P9U', '1Q2W', '1UJ1', '1UK2', '1UK3', '1UK4', '1WOF', '1Z1I']
```

These are the proteins containing the domain profile of coronavirus main protease (M-pro).

To find patterns in the domain, I created a similar variable handle, to which I passed in the ExPASy module followed by the prosite function, to which I passed in another ID PS00001. The prosite handle is then read.

In [56]:
```
handle = ExPASy.get_prosite_raw('PS00001')
record = Prosite.read(handle)
print(record.pattern)
```

```
N-{P}-[ST]-{P}.
```

The result shows that the common pattern within the domain is asparagine, proline, serine, and threonine.

---

5. KEGG

In this section, I deployed the KEGG module to procure genes and the pathways from the KEGG database using the enzyme commission (EC) number, which is assigned to every enzyme which has been studied or discovered so far.

Import Modules

In [57]:
```
from Bio.KEGG import REST, Enzyme
```

I created this variable named 'request' to fetch entries from the KEGG database. the 'kegg_get' function retrieves the KEGG molecule based on its EC number which is passed in as a string.

In [58]:
```
request = REST.kegg_get("ec:5.4.2.2")
open("ec_5.4.2.2.txt","w").write(request.read())
```

Out[58]:
```
264121
```

In this step, I fetched the enzyme classes present or associated with the EC number 5.4.2.2.

In [59]:
```
records = Enzyme.parse(open("ec_5.4.2.2.txt"))
record = list(records)[0]
record.classname
```

Out[59]:
```
['Isomerases;',
 'Intramolecular transferases;',
 'Phosphotransferases (phosphomutases)']
```

The pathways associated with the enzyme:

```
In [60]: record.pathway
```

```
Out[60]: [('PATH', 'ec00010', 'Glycolysis / Gluconeogenesis'),
          ('PATH', 'ec00030', 'Pentose phosphate pathway'),
          ('PATH', 'ec00052', 'Galactose metabolism'),
          ('PATH', 'ec00230', 'Purine metabolism'),
          ('PATH', 'ec00500', 'Starch and sucrose metabolism'),
          ('PATH', 'ec00520', 'Amino sugar and nucleotide sugar metabolism'),
          ('PATH', 'ec00521', 'Streptomycin biosynthesis'),
          ('PATH', 'ec01100', 'Metabolic pathways'),
          ('PATH', 'ec01110', 'Biosynthesis of secondary metabolites'),
          ('PATH', 'ec01120', 'Microbial metabolism in diverse environments')]
```

The (first ten) genes associated with these pathways:

```
In [61]: record.genes[:10]
```

```
Out[61]: [('HSA', ['5236', '55276']),
          ('PTR', ['456908', '461162']),
          ('PPS', ['100977295', '100993927']),
          ('GGO', ['101128874', '101131551']),
          ('PON', ['100190836', '100438793']),
          ('NLE', ['100596081', '100600656']),
          ('HMH', ['116456694', '116457795']),
          ('MCC', ['100424648', '699401']),
          ('MCF', ['101925921', '102130622']),
          ('MTHB', ['126935012', '126954887'])]
```

I fetched the genes involved by using a *for* loop, to eliminate the numbers involved:

```
In [62]: list_genes = []
         for x,y in record.genes:
             list_genes += x.split("\n")

         print(list_genes[:10])
```

```
['HSA', 'PTR', 'PPS', 'GGO', 'PON', 'NLE', 'HMH', 'MCC', 'MCF', 'MTHB']
```