INTRODUCTION: Python and Biopython-

Python stands as a truly versatile high-level programming language subjected to uses in scientific computing, data analysis, and automation. Because of its simple syntax, readability, and a massive library repertoire able to handle biological data that usually involves large sequence data with complicated analysis, it is a perfect choice. Python supports researchers in automating repetitive tasks, working through datasets programmatically, and building workflows that are reproducible.

Biopython is a custom-built Python application library used in the computation of biology and bioinformatics. The library consists of the tool sets that allow for handling biological sequences like DNA, RNA, or proteins; analysis of sequences; database queries; and integration with visualization tools. To ensure maximum effectiveness and reproducibility in larger-scale analysis, the interfaces that Biopython provides allow the researcher to program all aspects of sequence parsing, transcription, translation, alignments, and BLAST.

Pipeline Overview-

This is a detailed, modular pipeline written in Python with Biopython, aimed at executing an end-to-end analysis of protein and DNA sequences. The flow mimics an actual Biopython-based analysis through the incorporation of several necessary steps: sequence parsing, feature analysis, transcription and translation, alignment of sequences, visualization, genome annotation, and database search.

The DNA sequence under investigation in this study is the Hepatitis B virus S gene (HBV S gene), which is a region essential to the formation of the viral envelope and the study of vaccines. The protein sequences derived from translation are also examined using pairwise alignments and BLAST searches to determine homologous sequences and evaluate evolutionary conservation.

Key Objectives of the Pipeline-

1. Sequence Processing: Reading and preprocessing FASTA sequences for downstream analysis.
2. Central Dogma Simulation: Transcription from DNA to mRNA and translation to protein, emphasizing codon usage and amino acid content.
3. Feature Analysis: Computing GC content, amino acid frequencies, and creating sequence logos for visual interpretation.
4. Sequence Alignment: Aligning protein chains to find conserved regions and structural homology.
5. Database Integration: Querying protein sequences with NCBI BLASTp against public databases, finding homologs and functional information.
6. Visualization: Generating graphical displays of sequence characteristics, alignments, and BLAST outputs for easier interpretation of data.

By integrating these analyses, the pipeline presents an integrative means of Biopython-based sequence analysis that illustrates how different computational tools can be used in concert to investigate biological sequences from raw data to functional information. The workflow is centered on reproducibility and modularity to facilitate future extension to more sequences or experiments.

METHODOLOGY AND CODE EVOLUTION:

The bioinformatics pipeline was built in an iterative and systematic way. The goal was to develop a modular, end-to-end process that could query biological databases such as as NCBI, align sequences, model the central dogma, carry out feature analysis, parse DNA and protein sequences, and display the output. Proof-of-concept scripts for each job were the starting point to the process, and these were progressively augmented and refactored to build the final end-to-end pipeline.

1. Central Dogma Analysis

The fundamental guideline that traces the flow of genetic information in an organism is referred to as the Central Dogma of Molecular Biology. It describes the process of decoding genetic information in DNA by the proteins that do most of the biological activities in the cell.

Broadly, it has three main operations:

1. Replication
   - DNA can produce copies of itself.
   - It also facilitates the transfer of genetic information without any loss.
2. Transcription
   - The transcript of DNA is messenger RNA (mRNA).
   - In eukaryotes, this change goes from a very stable, long-term storage molecule (DNA) to a more active one (RNA) that can leave the nucleus.
3. Translation
   - The ribosome takes the data of mRNA and uses it tomakeprotein a chain of amino acids is read.
   - Proteins in the cell carry out different functions such as structure, catalysis (enzymatic), regulation, and signaling.

The whole process can be indicated like this:

DNA → RNA → Protein

The need for central dogma:

Central Dogma starts from the description of the flow of genetic information from DNA to RNA to proteins which ultimately links genotype and phenotype. It is a basic platform for understanding what goes on in cells, how mutations lead to diseases, and how changes in genetics can be used in biotech. It is the pillar of modern biology and life sciences as its concepts are applied in medicine (e.g., genetic disorders diagnosis, mRNA vaccine production), drug development, agriculture, and

Initial Script:

```python
# asmi chakraborty
# working with biopython
from Bio import SeqIO
# Read the FASTA file properly
record = SeqIO.read("sequence (2).fasta.txt", "fasta")
DNA = record.seq

print("This is the original DNA sequence:  ", DNA)
# Create a complement sequence
print("This is a complement sequence:      ", DNA.complement())
# Create a reverse complement sequence
print("This is a reverse complement seq:   ", DNA.reverse_complement())
# Create an mRNA sequence
print("This is an mRNA sequence:           ", DNA.transcribe())
# Create a protein sequence
print("This is a protein sequence:         ", DNA.translate())
```

```
 the original DNA sequence:
GCACAACATCAGGATTCCTAGGACCCCTGCTCGTGTTACAGGCGGGGTTTTTCTTGTTGACAAGAATCCTCACAATACCACAGAGTCTAGACTCGTGGTGGA
This is a complement sequence:
TACCTCTCGTGTTGTAGTCCTAAGGATCCTGGGGACGAGCACAATGTCCGCCCCAAAAAGAACAACTGTTCTTAGGAGTGTTATGGTGTCTCAGATCTGAGC
This is a reverse complement seq:
AATGTATACCCAAAGACAAAAGAAAATTGGTAATAGAGGTAAAAAGGGACTCAAGATGTTGTACAGACTTGGCCCCCAATACCACATCATCCATATAACTGAA
This is an mRNA sequence:
AUGGAGAGCACAACAUCAGGAUUCCUAGGACCCCUGCUCGUGUUACAGGCGGGGUUUUUCUUGUUGACAAGAAUCCUCACAAUACCACAGAGUCUAGACUCGU
This is a protein sequence:
MESTTSGFLGPLLVLQAGFFLLTRILTIPQSLDSWWTSLNFLGGAPTCPGQNSQSPTSNHSPTSCPPICPGYRWMCLRRFIIFLFILLLCLIFLLGLLDYQGM
```

+ Code + Markdown

Evolution and Integration:

- The initial script laid the groundwork for central dogma simulation and DNA parsing.
- For the final pipeline:
  - Automatically trims DNA sequences to multiples of three to allow for proper translation.
  - Automates transcription to mRNA and translation to protein and stores for downstream analysis.
  - Partial sequence outputs presented for verification and debugging.
  - Complement and reverse-complement functionality were preserved as part of sequence sanity checks.

Enhancements:

- Included calculation of GC content over a sliding window.
- Computationally integrated graphical visualization of GC distribution.
- Protein sequences are scanned for amino acid composition and sequence logos, giving extra biological information.

2. GC Content Analysis

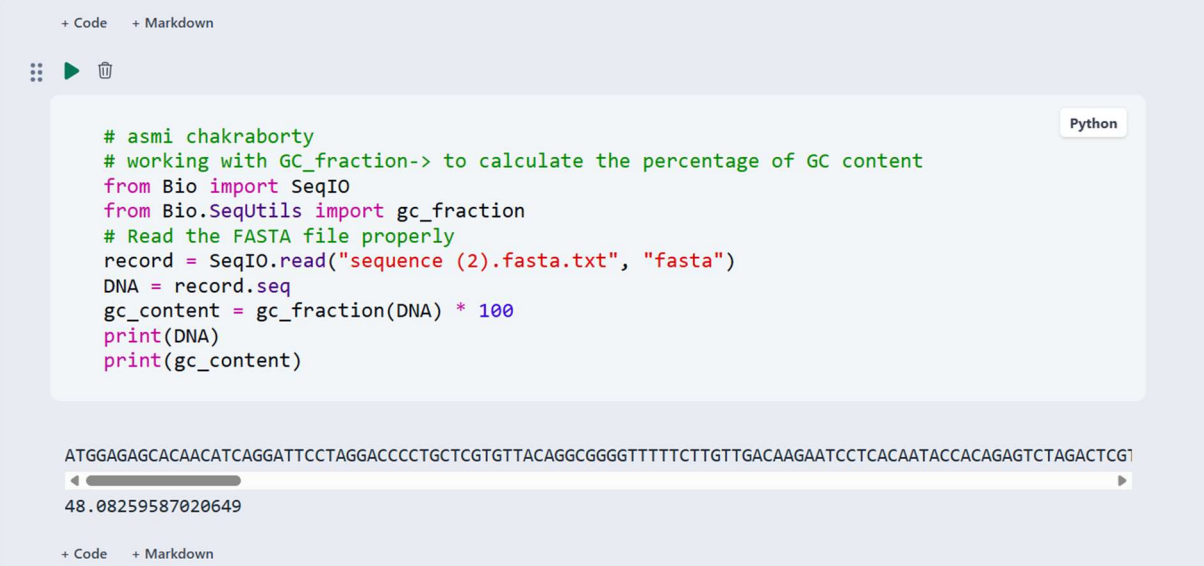
A proportion of guanine (G) and cytosine (C) bases in a DNA or RNA strand is called the GC content and it is calculated as:

$$GC\% = \frac{(G+C)}{(A+T+G+C)} \times 100$$

Importance of GC Content:

1. Genomic Stability: G–C rich sequences are more stable and denaturation-resistant as G–C pairs are stabilized by three hydrogen bonds compared to A–T pairs, which are guarded by two.
2. Species-Specific Marker: Many species exhibit characteristic GC content (e.g., human genome ~41% and E. coli ~50%), which is informative for comparative genomics and identification of taxonomy.
3. Gene expression regulation: Transcriptional function is frequently regulated by GC-rich regulatory elements and promoters.
4. Biotechnology Applications: Due to the fact that high GC content affects DNA stability and amplification, it is considered while designing synthetic genes, PCR primers, and cloning strategies.

Initial Script:



Evolution and Integration:

- Originally, the code calculated one GC percentage for the sequence.
- In the pipeline, it was expanded to a sliding window strategy:
  - The genome is read in 50 nucleotide windows (parameterizable), and GC content is calculated locally.
  - Plots GC content profile with GC-rich and AT-rich regions along the DNA sequence.

- Facilitates visual identification of genome regions with possible biological interest, i.e., promoters or coding regions.

Enhancements:

- Visualization was done using Matplotlib with reference lines for 50% GC content.
- Allows for comparative genomic analyses when generalized to multiple sequences.

3. Pairwise Sequence Alignment

Aligning sequences is a means to find common features between DNA, RNA, or protein sequences. The sequences can be related in terms of their structure, function, or evolution by these similarities.

Pairwise alignment is applied to two sequences, while multiple sequence alignment is for more than two sequences.

Importance of Sequence Alignment:

- Functional Insights: The parts that are conserved will often show important function areas or active sites of proteins.
- Annotation of Genomes: Helps the prediction of a new gene function analogizing with the already known sequences.
- Drug and Vaccine Design: Identifies those segments of the protein that can be targeted by drugs.
- Structural Prediction: Proteins which 3D structures are originated from homologous sequences can be simulated by using alignments.
- Mutant/ variant detection: Single nucleotide polymorphisms (SNPs), insertions, deletions, and mutations that cause diseases can all be found with the help of mutation/variant detection. .

Initial Script:

```python
#asmi chakraborty
#alignment: where two or more sequences are aligned together
from Bio import Align
from Bio.Seq import Seq
aligner = Align.PairwiseAligner()
aligner.mode = "global"
seq1 = "ATTCCTAAGT"
seq2 = "TCGT"
alignments = aligner.align(seq1,seq2)
print(alignments[0])
```

```
target            0 ATTCCTAAGT 10
                  0 -|-|----|| 10
query             0 -T-C----GT  4
```

Evolution and Integration:

- Originally applied to straightforward demonstration sequences.
- In development:
  - Used on actual protein sequences, namely Caspase-3 chains (2XYG, Chain A and B).
  - Global alignment enables comparison of whole protein sequences.
  - Graphical display of matches (1 = match, 0 = mismatch) emphasizes conserved motifs.

Enhancements:

- Alignment is now dynamic, taking any FASTA input sequences.
- Combined textual alignment output with match profile plots, improving interpretability.

4. BLAST Integration

The familiar BLAST (Basic Local Alignment Search Tool) algorithm is one of local similarity identification methods since it matches a query DNA or protein sequence with large databases by looking for short **regions of local similarity**

How BLAST Helps:

- Homology Detection: Makes easier the discovery of genes or proteins with similar functions by identifying homologous sequences in public databases.
- Functional annotation predicts they probably perform the functions that are assigned to them when they are linked to the already characterized sequences.
- Evolutionary Analysis: Finds conserved genes in all species to reveal the evolutionary relationships.
- Clinical Applications: Used in medical genomics for mutation analysis, diagnostic testing, and pathogen detection.
- Drug and vaccine research supports the identification of conserved targets between strains, thus, helping the creation of broad-spectrum therapies.
- Database Integration: Facilitates access to NCBI GenBank and PDB, two globally coordinated databases, for the purpose of reference comparisons that are always up-to-date.

The protein structure used here is 2XYG:

- 2XYG is a PDB code that stands for the crystal structure of the Hepatitis B Virus (HBV) small surface antigen (HBsAg), which is a major factor in viral assembly and infectivity.
- HBsAg is the main structural element of the HBV surface protein, hence the vaccines (such as the recombinant HBV vaccine) that target this structure are the most common sources of research.

Initial Script:

```
#asmi chakraborty
from Bio.Blast import NCBIWWW
from Bio.Blast import NCBIXML

# Define Query Sequences
protein_sequence = "MSCRSYRVVSSGHRVGNFSSCSAMTPQNLNRFRANSVSCWSGPGFRGLGSFGSRSVITFGSYSPRIAAVGSRPIHCGVRFGAGCGMGFGDGRGVGLGPRADSCVGLGFGAGSGIGYGFGGPGFGYRVGGVGVPAAPSITAVTVN

# Running BLASTp (Protein BLAST)
print("\nRunning blastp on database swissprot...")
result_handle_p = NCBIWWW.qblast("blastp", "swissprot", protein_sequence)
blast_record_p = NCBIXML.read(result_handle_p)

# Display top 3 alignments for BLASTp
print("\nTop 3 Alignments for blastp:")
for alignment in blast_record_p.alignments[:3]:
    print(f"\nSequence: {alignment.title}")
    print(f"Length: {alignment.length}")
    for hsp in alignment.hsps:
        print(f"Score: {hsp.score}, E-value: {hsp.expect}")
        print(f"Query: {hsp.query}")
        print(f"Match: {hsp.match}")
        print(f"Subject: {hsp.sbjct}")
```

```
Running blastp on database swissprot...

Top 3 Alignments for blastp:

Sequence: sp|Q9NSB2.2| RecName: Full=Keratin, type II cuticular Hb4; AltName: Full=Keratin-84; Short=K84; AltName: Full=Type II hair keratin Hb4; AltName: Full=Type
Length: 600
```

Evolution and Integration:

- Originally tested with a limited protein sequence.
- Pipeline:
  - Automated BLASTp queries are run on translated HBV protein sequences.
  - Results are saved as XML (blast_results.xml) for reproducibility.
  - Parsing creates a tabular summary with accession numbers, alignment scores, E-values, identities, and alignment lengths.
  - Top hits are represented as bar plots, displaying comparative bit scores.

Enhancements:

- Supports large sequences and avoids duplicate queries utilizing saved XML results.
- Supports swift identification of homologs and functional annotation.

5. Modular Pipeline Construction

Integration Strategy:

- Each of the original scripts was rewritten as a modular function or step:
- Central Dogma Module: DNA → mRNA → Protein.
  - Feature Analysis Module: GC content, amino acid composition, sequence logos.
  - Alignment Module: Pairwise protein alignment and match visualization.
  - BLAST Module: Automated querying, parsing, and plotting of results.
  - Automation: Dynamic reading of FASTA files; outputs (plots, XML, tables) are written to structured directories.
- Visualization: All steps have visual outputs for simple interpretation by non-experts.
- Reproducibility: XML and CSV outputs provide reproducible results across sessions.

6. Summary of Enhancements

Feature	Initial Code	Enhancement
DNA Parsing	Single sequence read	Any FASTA input, trimmed, verified
Central Dogma	Transcription & translation	Automated, stored, visualized
GC Content	Single percentage	Sliding window profile, plotted
Amino Acid Analysis	Not included	Frequency bar plots + sequence logos
Alignment	Hardcoded sequences	Dynamic protein alignment with match plots
BLAST	Single query	Automated BLASTp with parsing, table & plot outputs
Visualization	None	Comprehensive graphical outputs throughout
Modularity	Scripts independent	Fully integrated, reusable modules

BIOPYTHON WORKING PIPELINE IN GOOGLE COLAB: (cell-wise)

1. **Cell 1:**

```
#asmi chakraborty
#install dependencies

!pip install --quiet biopython matplotlib logomaker seaborn
print("Installed: biopython, matplotlib, logomaker, seaborn")
```

2. **Cell 2:**

```
#asmi chakraborty
#imports + set-up

import os
from datetime import datetime, timezone
import matplotlib.pyplot as plt
import seaborn as sns
import logomaker
import pandas as pd

from Bio import SeqIO, Align, Entrez
from Bio.Blast import NCBIWWW, NCBIXML
from Bio.SeqUtils import gc_fraction
from Bio.Seq import Seq
from collections import Counter

# Setup email for NCBI Entrez
Entrez.email = "asmichack@gmail.com"

def safe_mkdir(path):
    os.makedirs(path, exist_ok=True)
```

3. **Cell 3:**

```
#asmi chakraborty
#load FASTA files

# HBV sequence (DNA)
hbv_seq = (
"ATGGAGAGCACAACATCAGGATTCCTAGGACCCCTGCTCGTGTTACAGGCGGGGTTTTTCT
TGTTGACAA"
"GAATCCTCACAATACCACAGAGTCTAGACTCGTGGTGGACTTCTCTCAATTTTCTAGGGGG
AGCACCCAC"
"GTGTCCTGGCCAAAATTCGCAGTCCCCAACCTCCAATCACTCACCAACCTCTTGTCCTCCA
ATTTGTCCT"
"GGCTATCGCTGGATGTGTCTGCGGCGTTTTATCATATTCCTCTTCATCCTGCTTCTATGCCT
CATCTTCT"
"TGTTGGGTCTTCTGGACTACCAAGGTATGTTGCCCGTTTGTCCTCTACTTCCAGGATCATCA
ACTACCAG"
"CACGGGACCATGCAAAACCTGCACGATTCCTGCTCAAGGAACCTCTATGTTTCCCTCTTGT
TGCTGTACA"
"AAACCTTCGGACGGAAACTGCACTTGTATTCCCATCCCATCATCCTGGGCTTTCGCAAGAT
TCCTATGGG"
"AGTGGGCCTCAGTCCGTTTCTCCTGGCTCAGTTTACTAGTGCCATTTGTTCAGTGGTTCGTA
GGGCTTTC"
"CCCCACTGTTTGGCTTTCAGTTATATGGATGATGTGGTATTGGGGGCCAAGTCTGTACAAC
ATCTTGAGT"
"CCCTTTTTACCTCTATTACCAATTTTCTTTTGTCTTTGGGTATACATT"
)

with open("central_dogma.fasta", "w") as fh:
    fh.write(">HBV_S_gene\n")
    for i in range(0, len(hbv_seq), 70):
        fh.write(hbv_seq[i:i+70] + "\n")

# Caspase-3 (protein)
_2xyg = (
">Chain_A\n"
"SGISLDNSYKMDYPEMGLCIIINNKNFHKSTGMTSRSGTDVDAANLRETFRNLKYEVRNKNDL
TREEIVELMRDVSKEDHSKRSSFVCVLLSHGEEGIIFGTNGPVDLKKITNFFRGDRCRSLTGKPK
LFIIQACRGTELDCGIET\n"
">Chain_B\n"
"HKIPVEADFLYAYSTAPGYYSWRNSKDGSWFIQSLCAMLKQYADKLEFMHILTRVNRKVATE
FESFSFDATFHAKKQIPCIVSMLTKELYFYH\n"
)
with open("2xyg_chains.fasta", "w") as fh:
    fh.write(_2xyg)

print("FASTA files created: central_dogma.fasta, 2xyg_chains.fasta")
```

4. **Cell 4:**

```
#asmi chakraborty
#central dogma + gc plot

records = list(SeqIO.parse("central_dogma.fasta", "fasta"))
dna = records[0].seq
dna = dna[:len(dna) - len(dna)%3]  # trim

mrna = dna.transcribe()
protein = dna.translate()

print("DNA (first 60bp):", dna[:60])
print("mRNA (first 60nt):", mrna[:60])
print("Protein (first 60aa):", protein[:60])

# GC content profile
window = 50
gc_vals = [gc_fraction(dna[i:i+window])*100 for i in range(len(dna)-window)]

plt.figure(figsize=(10,4))
plt.plot(gc_vals, label="GC% (window=50)")
plt.axhline(50, color="red", linestyle="--", label="50% GC")
plt.xlabel("Position (bp)")
plt.ylabel("GC%")
plt.title("GC Content Profile")
plt.legend()
plt.show()
```

5. **Cell 5:**

```
#asmi chakraborty
#amino acid frequency + sequence logo

# 1. Amino acid frequency
aa_counts = Counter(str(protein))
plt.figure(figsize=(10,4))
sns.barplot(x=list(aa_counts.keys()), y=list(aa_counts.values()), color="skyblue")
plt.title("Amino Acid Composition")
plt.xlabel("Amino Acid")
plt.ylabel("Count")
plt.show()

# 2. Sequence Logo (first 30 aa)
logo_seq = str(protein)[:30]
counts_df = logomaker.alignment_to_matrix([logo_seq], to_type="counts")

plt.figure(figsize=(10,3))
logomaker.Logo(counts_df)
plt.title("Sequence Logo (first 30 aa)")
plt.show()
```

6. **Cell 6:**

```
#asmi chakraborty
#pairwise alignment visualization

chains = list(SeqIO.parse("2xyg_chains.fasta", "fasta"))
seq1, seq2 = chains[0].seq, chains[1].seq

aligner = Align.PairwiseAligner()
aligner.mode = "global"
alignment = aligner.align(seq1, seq2)[0]

print(alignment)

# Match profile
match_profile = [1 if a==b else 0 for a,b in zip(str(seq1[:len(seq2)]), str(seq2))]
plt.figure(figsize=(10,3))
plt.plot(match_profile, drawstyle="steps-mid")
plt.title("Alignment Match Profile (Chain A vs B)")
plt.xlabel("Position")
plt.ylabel("Match (1=yes,0=no)")
plt.show()
```

7. **Cell 7:**

```
#asmi chakraborty
#run BLASTp on protein

print("⏳Running BLASTp on HBV protein sequence...")
result_handle = NCBIWWW.qblast("blastp", "nr", protein)

# Save raw results
with open("blast_results.xml", "w") as out_handle:
    out_handle.write(result_handle.read())
result_handle.close()
print(" BLAST search completed and saved to blast_results.xml")
```

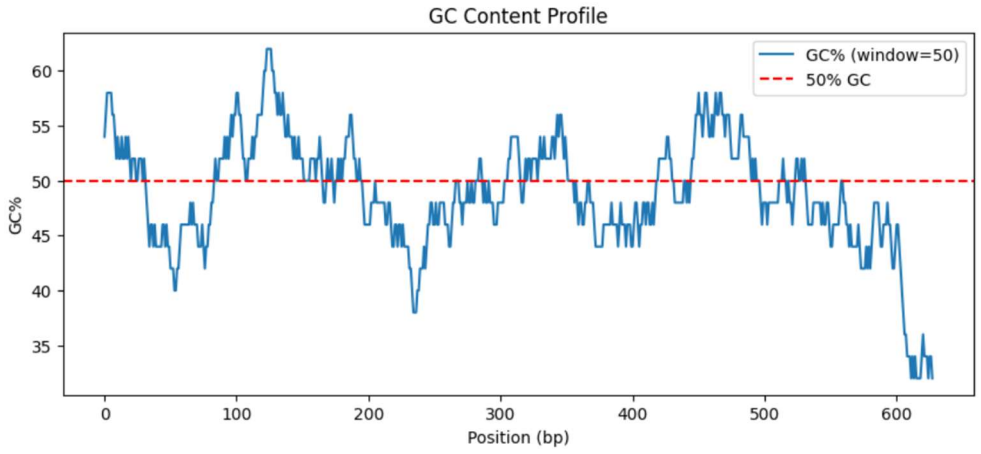
8. **Cell 8:**

```
#asmi chakraborty
#parse BLAST result

blast_records = NCBIXML.parse(open("blast_results.xml"))
blast_record = next(blast_records)

# Top 5 hits summary
hits = []
for alignment in blast_record.alignments[:5]:
    for hsp in alignment.hsps:
        hits.append({
            "Accession": alignment.accession,
            "Title": alignment.title[:80]+"...",
            "Score": hsp.score,
            "E-value": hsp.expect,
            "Identity": hsp.identities,
            "Align Length": hsp.align_length
        })

df_hits = pd.DataFrame(hits)
print(df_hits)

# Barplot of bit scores
plt.figure(figsize=(8,4))
sns.barplot(x="Accession", y="Score", data=df_hits, color="lightgreen")
plt.title("Top BLASTp Hits (Bit Scores)")
plt.xticks(rotation=45)
plt.show()
```

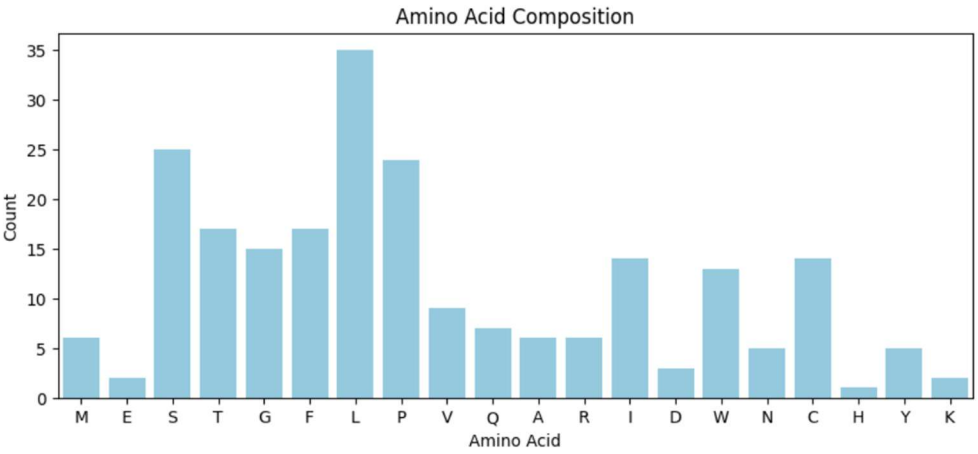
RESULTS OBSERVED:

1. For central dogma and gc fraction:

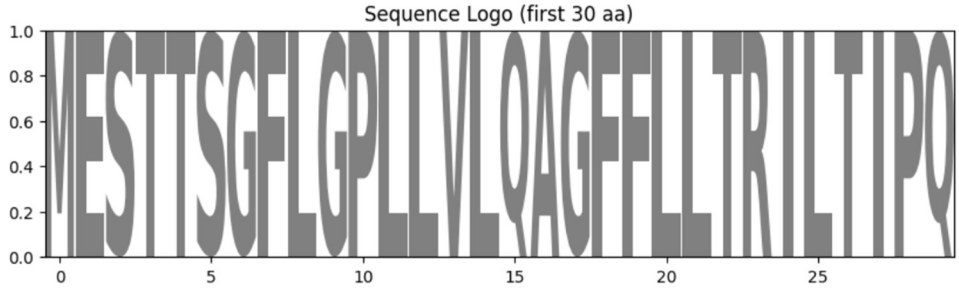
```
DNA (first 60bp): ATGGAGAGCACAACATCAGGATTCCTAGGACCCCTGCTCGTGTTACAGGCGGGGTTTTTC
mRNA (first 60nt): AUGGAGAGCACAACAUCAGGAUUCCUAGGACCCCUGCUCGUGUUACAGGCGGGGUUUUUC
Protein (first 60aa): MESTTSGFLGPLLVLQAGFFLLTRILTIPQSLDSWWTSLNFLGGAPTCPGQNSQSPTSNH
```



2. For amino acid composition and sequence logo of first 30 amino acids:

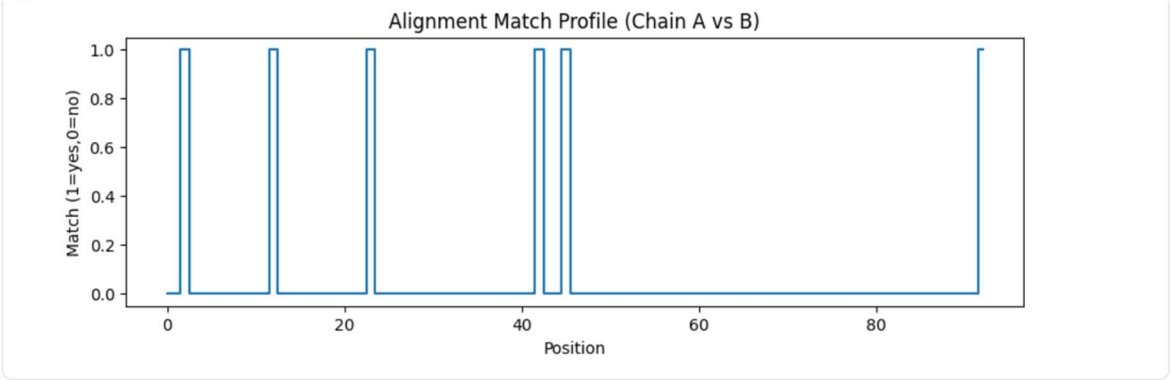



```
<Figure size 1000x300 with 0 Axes>
```



3. For pairwise alignment visualization:

```
target          0 SGISLDNSY-KMDY-P-EMGLCIIINNKN--FHK----ST--GMT--S-R-SGT-DVDAA
                0 ----------|----|-|-------------|------||--|----|-|-|---|----
query           0 ---------HK---IPVE-----------ADF--LYAYSTAPG--YYSWRNS--KD----

target         44 NLRET---FRN---L----K-YEVRNKN-D-LTREE---IVELM-RDVS--K---EDH--
               60 --------|-----|----|-|-------|-|--|----|--|--|-|---|---|----
query          27 -----GSWF--IQSLCAMLKQY------ADKL--E-FMHI--L-TR-V-NRKVATE--FE

target         80 SKRSSFVCVLLSHGEEGIIFGTNGPVDL-----KK-ITNFFRGDR-CR--S-LTGKPKLF
              120 |----|-----|-------|------|------||-|---------|---|-||-|----
query          64 S----F-----S-------F------D-ATFHAKKQI--------PC-IVSMLT-K----

target        130 IIQACRGTELDCGIET---- 146
              180 --------||---------- 200
query          87 --------EL------YFYH  93
```

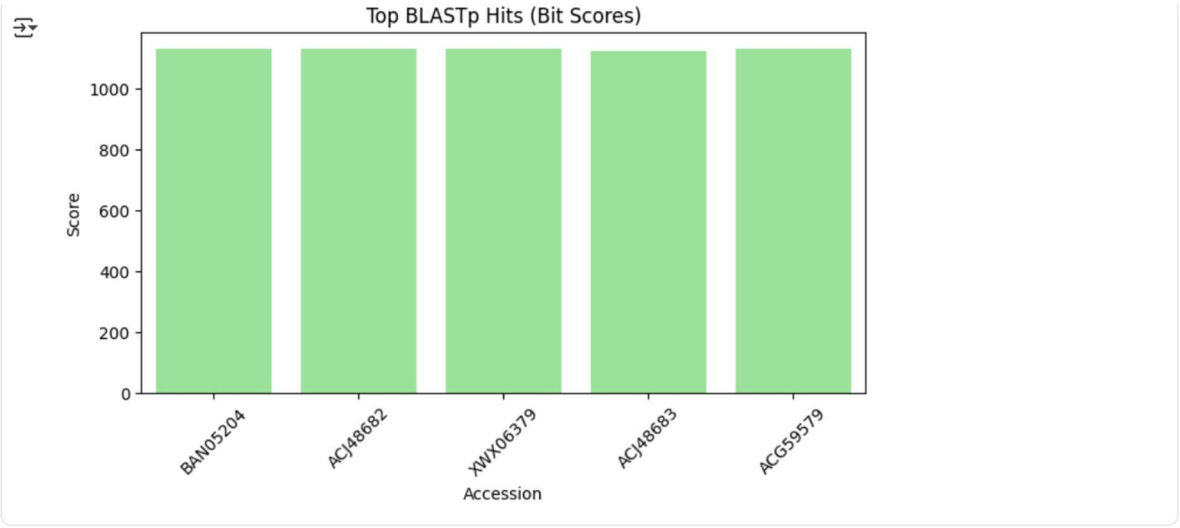


4. For BLASTp on protein


```
<BlastOutput>
  <BlastOutput_program>blastp</BlastOutput_program>
  <BlastOutput_version>BLASTP 2.17.0+</BlastOutput_version>
  <BlastOutput_reference>Stephen F. Altschul, Thomas L. Madden, Alejandro A. Sch&auml;ffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997), "Gapped BLAST and PSI-BLAST: a
  new generation of protein database search programs", Nucleic Acids Res. 25:3389-3402.</BlastOutput_reference>
  <BlastOutput_db>nr</BlastOutput_db>
  <BlastOutput_query-ID>Query_168849</BlastOutput_query-ID>
  <BlastOutput_query-def>unnamed protein product</BlastOutput_query-def>
  <BlastOutput_query-len>226</BlastOutput_query-len>
  <BlastOutput_param>
    <Parameters>
      <Parameters_matrix>BLOSUM62</Parameters_matrix>
      <Parameters_expect>10</Parameters_expect>
      <Parameters_gap-open>11</Parameters_gap-open>
      <Parameters_gap-extend>1</Parameters_gap-extend>
      <Parameters_filter>F</Parameters_filter>
    </Parameters>
  </BlastOutput_param>
  <BlastOutput_iterations>
    <Iteration>
      <Iteration_iter-num>1</Iteration_iter-num>
      <Iteration_query-ID>Query_168849</Iteration_query-ID>
      <Iteration_query-def>unnamed protein product</Iteration_query-def>
      <Iteration_query-len>226</Iteration_query-len>
      <Iteration_hits>
        <Hit>
          <Hit_num>1</Hit_num>
          <Hit_id>dbj|BAN05204.1|</Hit_id>
          <Hit_def>S, partial [Hepatitis B virus]</Hit_def>
          <Hit_accession>BAN05204</Hit_accession>
          <Hit_len>226</Hit_len>
          <Hit_hsps>
            <Hsp>
              <Hsp_num>1</Hsp_num>
              <Hsp_bit-score>439.113</Hsp_bit-score>
              <Hsp_score>1128</Hsp_score>
              <Hsp_evalue>1.12173e-154</Hsp_evalue>
              <Hsp_query-from>1</Hsp_query-from>
              <Hsp_query-to>226</Hsp_query-to>
              <Hsp_hit-from>1</Hsp_hit-from>
              <Hsp_hit-to>226</Hsp_hit-to>
              <Hsp_query-frame>0</Hsp_query-frame>
              <Hsp_hit-frame>0</Hsp_hit-frame>
              <Hsp_identity>226</Hsp_identity>
```

5. For parse BLAST result

```
  Accession                                              Title   Score  \
0  BAN05204  dbj|BAN05204.1| S, partial [Hepatitis B virus]...  1128.0
1  ACJ48682  gb|ACJ48682.1| middle S protein [Hepatitis B v...  1129.0
2  XWX06379  gb|XWX06379.1| middle S protein [Hepatitis B v...  1128.0
3  ACJ48683  gb|ACJ48683.1| S protein [Hepatitis B virus] >...  1122.0
4  ACG59579  gb|ACG59579.1| middle S protein [Hepatitis B v...  1128.0

         E-value  Identity  Align Length
0  1.121730e-154       226           226
1  5.409650e-154       225           226
2  8.909470e-154       224           226
3  9.218520e-154       225           226
4  9.726210e-154       224           226
```



STEP-WISE EXPLANATION OF THE BIOPYTHON PIPELINE:

1. **Cell 1: Install Dependencies-**

Code used:

```
!pip install --quiet biopython matplotlib logomaker seaborn
```

Explanation:

- biopython: Central bioinformatics package for sequence parsing, translation, BLAST, and alignments.
- matplotlib: Default plotting package for bar/line plots.
- seaborn: High-level plotting (pretty barplots, heatmaps).
- logomaker: Produces sequence logos to visualize conserved amino acids.
- --quiet: prevents filling output with installation messages.

Purpose: Guarantees that all necessary packages are installed prior to running the pipeline.

2. **Cell 2: Imports + Setup**

Code used:

```
from Bio import SeqIO, Align, Entrez
from Bio.Blast import NCBIWWW, NCBIXML
from Bio.Seq import Seq
from Bio.SeqUtils import gc_fraction
import matplotlib.pyplot as plt
import seaborn as sns
import logomaker
from collections import Counter
from datetime import datetime, timezone
import os
import pandas as pd

Entrez.email = "asmichack@gmail.com"
```

Explanation:

- SeqIO: Read/write FASTA/GenBank files.
- Align: Perform pairwise sequence alignments.
- Entrez: Access NCBI databases (not fully used here, but good for future expansion).
- NCBIWWW.qblast, NCBIXML: Run online BLAST searches and parse XML results.
- gc_fraction: Calculate GC content of DNA.
- Counter: Count amino acids for frequency plots.
- timezone & datetime: For generating timestamped reports.
- safe_mkdir: Function to create folders for storing outputs.

Purpose: Sets up all tools, plotting libraries, and NCBI email (required for BLAST queries).

3. **Cell 3: Load FASTA Files**

Code used:

```
with open("central_dogma.fasta", "w") as fh: ...
with open("2xyg_chains.fasta", "w") as fh: ...
```

Explanation:

- Creates FASTA files from sequences you provided.
- central_dogma.fasta: HBV S-gene DNA sequence.
- 2xyg_chains.fasta: Caspase-3 chains (protein).
- Splits the DNA sequence every 70 bases (FASTA convention).

Purpose: Prepares input files for the pipeline — DNA for central dogma, protein for alignment & BLAST.

4. **Cell 4: Central Dogma + GC Plot**

Code used:

```
records = list(SeqIO.parse("central_dogma.fasta", "fasta"))
dna = records[0].seq
dna = dna[:len(dna) - len(dna)%3]
mrna = dna.transcribe()
protein = dna.translate()
```

Step-by-step:

1. SeqIO.parse: Reads FASTA file.
2. dna[:len(dna) - len(dna)%3]: Trims DNA to a multiple of 3 (codons).
3. .transcribe(): Converts DNA → mRNA (T → U).
4. .translate(): Converts DNA → protein using codons.

Visualization:

```
gc_vals = [gc_fraction(dna[i:i+window])*100 ...]
plt.plot(gc_vals)
```

- Sliding window GC content: Shows local GC-rich or AT-rich regions.
- Red line at 50%: Reference line.

Purpose: Simulates central dogma and shows GC distribution, useful in gene analysis.

5. **Cell 5: Amino Acid Frequency + Sequence Logo**

Code used:

```
aa_counts = Counter(str(protein))
sns.barplot(x=list(aa_counts.keys()), y=list(aa_counts.values()))
```

Explanation:

- Counter: Counts how many times each amino acid appears.
- sns.barplot: Bar plot for visualization.

Sequence Logo:

```
counts_df = logomaker.alignment_to_matrix([logo_seq], to_type="counts")
logomaker.Logo(counts_df)
```

- Shows conserved amino acids in the first 30 residues of the protein.
- Taller letters → more conserved positions.

Purpose: Visualize composition and conservation of the translated protein.

6. **Cell 6: Pairwise Alignment Visualization**

Code used:

```
aligner = Align.PairwiseAligner()
aligner.mode = "global"
alignment = aligner.align(seq1, seq2)[0]
```

Explanation:

- PairwiseAligner: Aligns two protein sequences.
- global: Full-length alignment.
- match_profile = [1 if a==b else 0 ...]: Creates a simple match profile plot (1 = match, 0 = mismatch).

Purpose: Shows similarity between Chain A & B of Caspase-3.

7. **Cell 7: Run BLASTp**

Code used:

```
result_handle = NCBIWWW.qblast("blastp", "nr", protein)
```

Explanation:

- qblast("blastp", "nr", protein): Runs protein BLAST against NCBI's non-redundant database.
- Saves results as XML for parsing: blast_results.xml.
- Might take several minutes (internet required).

Purpose: Finds homologous sequences for your protein.

Cell 8: Parse BLAST Results

Code used:

```
blast_records = NCBIXML.parse(open("blast_results.xml"))
```

Step-by-step:

- NCBIXML.parse: Parses the BLAST XML file.
- blast_record.alignments[:5]: Top 5 hits.
- Extract score, e-value, identity, alignment length.
- Converts to Pandas DataFrame → easier visualization.

Visualization:

```
sns.barplot(x="Accession", y="Score", data=df_hits)
```

- Shows bit scores of top BLAST hits.
- Quickly identifies the most similar sequences.

Purpose: Turns BLAST results into easy-to-read table + plot for downstream analysis.

Summary of the Pipeline

Step	Input	Output	Visualization
1.	DNA FASTA	mRNA + Protein	GC content line plot
2.	Protein	Amino acid counts	AA frequency bar plot
3.	Protein	First 30 amino acids	Sequence logo
4.	Caspase chains	Alignment	Match profile plot
5.	Protein	BLAST hits	Table + bit score bar plot

REAL LIFE APPLICATIONS:

The modular Biopython pipeline created in this work finds applications in various fields of molecular biology, computational biology, medicine, and biotechnology. The major applications are as follows:

1. Viral Genomics and Vaccine Development
    - The pipeline can be used to examine viral DNA and protein sequences like the Hepatitis B virus S gene employed in this work.
    - Knowledge of the GC content, codon usage, and amino acid composition is vital in synthetic gene design and codon optimization, which are compulsory in vaccine development and recombinant protein production.
    - Protein sequence analysis and BLAST can predict conserved epitopes to facilitate antigen design for vaccines.
2. Protein Functional Annotation and Homology Analysis
    - Scientists can identify homologous sequences in public databases that provide information on the protein structure, function, and evolutionary conservation through the application of BLASTp searches.
    - Finding drug targets or enzyme catalysis relies on finding conserved motifs, active sites, and binding regions, which are facilitated by pairwise and multiple sequence alignments.
3. Drug Target Identification and Therapeutic Research
    - By comparing proteins of pathogens or disease genes, the pipeline can pinpoint potential drug targets.
    - Conserved areas identified with alignment and BLAST are best candidates for small molecule inhibitors or antibody development.
4. Comparative Genomics and Evolutionary Biology
    - Genes from multiple strains or species can be compared through GC content profiles, amino acid composition analysis, and sequence logos to uncover evolutionary tendencies and hotspot mutations.
    - It assists in gene conservation research, strain-specific virulence, and monitoring viral evolution.
5. Training and Educational Use
    - The pipeline is a hands-on learning tool for Biopython, sequence analysis, and database integration.
    - Early researchers and students can programmatically run simulations of the central dogma, investigate sequence properties, visualize data, and run BLAST queries.
    - Visualization modules like GC plots, amino acid frequency plots, and sequence logos facilitate better understanding and interpretation of biological data.
6. High-Throughput Genomic Studies
    - The scalability and modularity of the pipeline enable it to be used to analyze large collections of genomic or proteomic sequences.
    - It can also be modified to screen numerous viral genomes or protein families at one time, which would facilitate easier molecular diagnosis, vaccine development, and genomics studies.

CONCLUSION:

The development of a robust, modular pipeline on top of Biopython that can handle end-to-end DNA and protein sequence analysis is illustrated in this work. The pipeline offers a repeatable, efficient, and scalable platform for biological sequence research by combining various functionalities, such as sequence parsing, central dogma simulation, calculation of GC content, amino acid analysis, sequence alignment, BLAST querying, and visualization.
Key Insights and Achievements:

1. Automation and Reproducibility:
    - The pipeline minimizes human error and generates uniform results by automating repetitive sequence analysis procedures.

2. Extensive Visualization:
    - Sliding window GC content plots, amino acid frequency bar plots, sequence logos, and alignment match profiles render the sequence data more understandable to readily discern important patterns and motifs.

3. Function and Evolution Insights:
    - Functional annotation, protein characterization, and evolutionary analysis are facilitated by the rapid access to homologous sequences offered by BLAST integration.

4. Modularity and Scalability:
    - Each module is independent and may be extended or replaced to deal with new types of analyses, different organisms, or bigger data without redefinition of the pipeline.

5. Bridging Computational Tools and Biology:
    - The pipeline illustrates how Python and Biopython can be employed to convert raw sequencing data into biological understanding that can be used to apply to comparative genomics, vaccine development, medicine, and teaching.

Finally, the pipeline established here provides an efficient process from raw DNA sequence data to functional protein analysis and interrogation of databases and is thus a very useful resource for students and researchers. Its modularity, visualization features, and integration with databases make real biological applications such as viral genomics, vaccine design, protein functional annotation, drug target discovery, and comparative genomics research feasible. The pipeline illustrates the real-world application of Python and Biopython in modern molecular biology research through the integration of computational power with biological relevance.