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-

In this study, we will showcase a full, modular Python pipeline built with Biopython to complete the analysis of a DNA and protein sequence from start to finish. This workflow represents a bioinformatics workflow that one might encounter in real life, with extra stages of analyses. The pipeline includes several different facets: (1) parsing a sequence, (2) applying the central dogma, (3) calculating DNA, RNA, and protein features, (4) profiling codon usage, (5) evaluating physicochemical properties of proteins, (6) visualizing protein structure, (7) performing a BLASTp search, and (8) reconstructing a phylogenetic tree.

For the purposes of this exercise, we have used the DNA sequence of the Hepatitis B Virus (HBV) S gene. This region encodes a highly coded protein and is an essential surface antigen and important target for vaccine development. In addition, translated protein sequences were evaluated by way of pairwise alignment, compositional analysis, and BLASTp to find homologous proteins and estimate evolutionary conservation.

Key Objectives of the Pipeline-

- Simulation of the Central Dogma: Execute transcription to examine the frequency of codon evidences, GC content and reading frames.
- Element and composition: Calculations to establish the percent GC, codon bias, amino acids, molecular weight, isoelectric point, aromaticity and instability index.
- Visualization: Create plots that add meaning and significance to visual artifacts such as GC content profiles, codon use, amino acid composition, hydropathy plots, and sequence logos to represent structural definitions.
- Sequence alignment: Leverage pairwise alignment to compile similar situated regions, or overall sequence similarities, between variants.
- Incorporation of BLASTp: Query translated proteins to search for translation against proteins classified by NCBI (nr) non-redundant database to catalogue homologous proteins and infer functional annotations.
- Structural and evolutionary: Recovery of 3-D models that represent proteins in RCSB PDB and visualizing in py3Dmol. Phylogenetic trees to represent evolutionary relationship distributions.

BIOPYTHON WORKING PIPELINE IN GOOGLE COLAB:

```
#asmi chakraborty
# installing dependencies
!pip install --quiet biopython matplotlib seaborn logomaker py3Dmol
pandas numpy scipy

# imports and setup
import os
import matplotlib.pyplot as plt
import seaborn as sns
import logomaker
import pandas as pd
import numpy as np
from collections import Counter
import py3Dmol
from IPython.display import display

from Bio import SeqIO, Align, Entrez, Phylo
from Bio.Blast import NCBIWWW, NCBIXML
from Bio.SeqUtils import gc_fraction, molecular_weight
from Bio.Seq import Seq
from Bio.PDB import PDBParser, PDBIO, DSSP
from Bio.SeqUtils.ProtParam import ProteinAnalysis

#setting up
Entrez.email = "asmichack@gmail.com"
sns.set_style("whitegrid")
print("✓ All dependencies imported successfully!")

#FASTA file creation (HBV DNA + Caspase-3 Protein)

hbv_seq = (
"ATGGAGAGCACAACATCAGGATTCCTAGGACCCCTGCTCGTGTTACAGGCGGGGTTTTTCTTGTTGACAA
"
"GAATCCTCACAATACCACAGAGTCTAGACTCGTGGTGGACTTCTCTCAATTTTCTAGGGGGAGCACCCAC
"
"GTGTCCTGGCCAAAATTCGCAGTCCCCAACCTCCAATCACTCACCAACCTCTTGTCCTCCAATTTGTCCT
"
"GGCTATCGCTGGATGTGTCTGCGGCGTTTTATCATATTCCTCTTCATCCTGCTTCTATGCCTCATCTTCT
"
"TGTTGGGTCTTCTGGACTACCAAGGTATGTTGCCCGTTTGTCCTCTACTTCCAGGATCATCAACTACCAG
"
"CACGGGACCATGCAAAACCTGCACGATTCCTGCTCAAGGAACCTCTATGTTTCCCTCTTGTTGCTGTACA
"
"AAACCTTCGGACGGAAACTGCACTTGTATTCCCATCCCATCATCCTGGGCTTTCGCAAGATTCCTATGGG
"
"AGTGGGCCTCAGTCCGTTTCTCCTGGCTCAGTTTACTAGTGCCATTTGTTCAGTGGTTCGTAGGGCTTTC
"
```

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

_2xyg = (
">Chain_A\n"
"SGISLDNSYKMDYPEMGLCIIINNKNFHKSTGMTSRSGTDVDAANLRETFRNLKYEVRNKNDLTREEIVE
LMRDVSKEDHSKRSSFVCVLLSHGEEGIIFGTNGPVDLKKITNFFRGDRCRSLTGKPKLFIIQACRGTELD
CGIET\n"
">Chain_B\n"
"HKIPVEADFLYAYSTAPGYYSWRNSKDGSWFIQSLCAMLKQYADKLEFMHILTRVNRKVATEFESFSFDA
TFHAKKQIPCIVSMLTKELYFYH\n"
)
with open("2xyg_chains.fasta", "w") as f:
    f.write(_2xyg)

print("✓ FASTA files created successfully!")

# central dogma analysis

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()

print("\n" + "="*60)
print("=== CENTRAL DOGMA ANALYSIS ===")
print("="*60)
print(f"DNA Length: {len(dna)} bp")
print(f"Protein Length: {len(protein)} aa")
print(f"Molecular Weight: {molecular_weight(protein, 'protein'):.2f}
Da")
print(f"\nDNA (first 60bp):\n{dna[:60]}")
print(f"\nmRNA (first 60nt):\n{mrna[:60]}")
print(f"\nProtein (first 60aa):\n{protein[:60]}")

#GC content analysis
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# GC content profile
window = 50
```

```
gc_vals = [gc_fraction(dna[i:i+window])*100 for i in range(len(dna)-
window)]
axes[0,0].plot(gc_vals, color='#2E86AB', linewidth=2)
axes[0,0].axhline(50, color='#A23B72', linestyle='--', linewidth=2,
label='50% GC')
axes[0,0].fill_between(range(len(gc_vals)), gc_vals, alpha=0.3,
color='#2E86AB')
axes[0,0].set_xlabel('Position (bp)', fontsize=11)
axes[0,0].set_ylabel('GC%', fontsize=11)
axes[0,0].set_title('GC Content Profile (Window=50bp)', fontsize=12,
fontweight='bold')
axes[0,0].legend()
axes[0,0].grid(alpha=0.3)

# nucleotide composition
nt_counts = {'A': dna.count('A'), 'T': dna.count('T'),
             'G': dna.count('G'), 'C': dna.count('C')}
colors = ['#F18F01', '#C73E1D', '#6A994E', '#BC4B51']
axes[0,1].bar(nt_counts.keys(), nt_counts.values(), color=colors,
edgecolor='black', linewidth=1.5)
axes[0,1].set_title('Nucleotide Composition', fontsize=12,
fontweight='bold')
axes[0,1].set_xlabel('Nucleotide', fontsize=11)
axes[0,1].set_ylabel('Count', fontsize=11)
axes[0,1].grid(axis='y', alpha=0.3)

# codon Usage (top 15)
codons = [str(dna[i:i+3]) for i in range(0, len(dna), 3)]
codon_counts = Counter(codons).most_common(15)
codon_names, codon_vals = zip(*codon_counts)
axes[1,0].barh(codon_names, codon_vals, color='#457B9D',
edgecolor='black')
axes[1,0].set_title('Top 15 Codon Usage', fontsize=12,
fontweight='bold')
axes[1,0].set_xlabel('Frequency', fontsize=11)
axes[1,0].invert_yaxis()
axes[1,0].grid(axis='x', alpha=0.3)

# reading frame GC content
frames_gc = []
for frame in range(3):
    frame_seq = dna[frame:]
    frame_seq = frame_seq[:len(frame_seq) - len(frame_seq)%3]
    frames_gc.append(gc_fraction(frame_seq)*100)

axes[1,1].bar(['Frame 0', 'Frame 1', 'Frame 2'], frames_gc,
              color=['#E63946', '#F1FAEE', '#A8DADC'],
edgecolor='black', linewidth=1.5)
```

```
axes[1,1].axhline(50, color='gray', linestyle='--', linewidth=2)
axes[1,1].set_title('GC Content by Reading Frame', fontsize=12,
fontweight='bold')
axes[1,1].set_ylabel('GC%', fontsize=11)
axes[1,1].grid(axis='y', alpha=0.3)

plt.tight_layout()
plt.show()

# protein analysis
analyzer = ProteinAnalysis(str(protein))

print("\n" + "="*60)
print("=== PROTEIN PROPERTIES ===")
print("="*60)
print(f"Molecular Weight: {analyzer.molecular_weight():.2f} Da")
print(f"Isoelectric Point: {analyzer.isoelectric_point():.2f}")
print(f"Aromaticity: {analyzer.aromaticity():.4f}")
print(f"Instability Index: {analyzer.instability_index():.2f}")
print(f"Gravy (Hydropathy): {analyzer.gravy():.4f}")

# secondary structure prediction
ss_frac = analyzer.secondary_structure_fraction()
print(f"\nSecondary Structure Prediction:")
print(f"  Helix: {ss_frac[0]:.2%}")
print(f"  Turn: {ss_frac[1]:.2%}")
print(f"  Sheet: {ss_frac[2]:.2%}")

#amino acid composition visualization
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

#amino acid frequency
aa_counts = Counter(str(protein))
sorted_aa = sorted(aa_counts.items(), key=lambda x: x[1], reverse=True)
aa_names, aa_vals = zip(*sorted_aa)

axes[0,0].bar(aa_names, aa_vals, color='#06FFA5', edgecolor='black',
linewidth=1.2)
axes[0,0].set_title('Amino Acid Composition', fontsize=12,
fontweight='bold')
axes[0,0].set_xlabel('Amino Acid', fontsize=11)
axes[0,0].set_ylabel('Count', fontsize=11)
axes[0,0].grid(axis='y', alpha=0.3)

#sequence logo
logo_seq = str(protein)[:30]
counts_df = logomaker.alignment_to_matrix([logo_seq], to_type='counts')
logomaker.Logo(counts_df, ax=axes[0,1])
```

```
axes[0,1].set_title('Sequence Logo (First 30 Residues)', fontsize=12,
fontweight='bold')

# hydropathy plot (kyte-doolittle)
window_size = 9
hydropathy_scores = analyzer.protein_scale(window=window_size,
param_dict={
    'A': 1.8, 'R': -4.5, 'N': -3.5, 'D': -3.5, 'C': 2.5,
    'Q': -3.5, 'E': -3.5, 'G': -0.4, 'H': -3.2, 'I': 4.5,
    'L': 3.8, 'K': -3.9, 'M': 1.9, 'F': 2.8, 'P': -1.6,
    'S': -0.8, 'T': -0.7, 'W': -0.9, 'Y': -1.3, 'V': 4.2
})
axes[1,0].plot(hydropathy_scores, color='#FF006E', linewidth=2)
axes[1,0].axhline(0, color='black', linestyle='--', linewidth=1)
axes[1,0].fill_between(range(len(hydropathy_scores)),
hydropathy_scores,
                       where=[h > 0 for h in hydropathy_scores],
                       alpha=0.3, color='#FF006E',
label='Hydrophobic')
axes[1,0].fill_between(range(len(hydropathy_scores)),
hydropathy_scores,
                       where=[h < 0 for h in hydropathy_scores],
                       alpha=0.3, color='#3A86FF',
label='Hydrophilic')
axes[1,0].set_title(f'Hydropathy Plot (Kyte-Doolittle,
Window={window_size})',
                    fontsize=12, fontweight='bold')
axes[1,0].set_xlabel('Position', fontsize=11)
axes[1,0].set_ylabel('Hydropathy Score', fontsize=11)
axes[1,0].legend()
axes[1,0].grid(alpha=0.3)

# secondary structure pie chart
ss_labels = ['Helix', 'Turn', 'Sheet']
ss_colors = ['#8338EC', '#FF006E', '#FFBE0B']
axes[1,1].pie(ss_frac, labels=ss_labels, colors=ss_colors,
autopct='%1.1f%%',
              startangle=90, textprops={'fontsize': 11, 'weight':
'bold'})
axes[1,1].set_title('Predicted Secondary Structure', fontsize=12,
fontweight='bold')

plt.tight_layout()
plt.show()

# enhanced pairwise alignment

chains = list(SeqIO.parse("/content/2xyg_chains.fasta", "fasta"))
```

```
seq1, seq2 = chains[0].seq, chains[1].seq

aligner = Align.PairwiseAligner()
aligner.mode = "global"
aligner.match_score = 2
aligner.mismatch_score = -1
aligner.open_gap_score = -2
aligner.extend_gap_score = -0.5

alignment = aligner.align(seq1, seq2)[0]

print("\n" + "="*60)
print("=== PAIRWISE ALIGNMENT (Chain A vs Chain B) ===")
print("="*60)
print(f"Alignment Score: {alignment.score}")
print(f"Length Chain A: {len(seq1)}")
print(f"Length Chain B: {len(seq2)}")
print("\n" + str(alignment)[:500] + "...")

# alignment visualization
min_len = min(len(seq1), len(seq2))
match_profile = [1 if seq1[i] == seq2[i] else 0 for i in
range(min_len)]
identity = sum(match_profile) / min_len * 100

fig, axes = plt.subplots(2, 1, figsize=(14, 8))

# match profile
axes[0].fill_between(range(min_len), match_profile, step='mid',
                     color='#06FFA5', alpha=0.6, label='Match')
axes[0].plot(match_profile, drawstyle='steps-mid', color='#023047',
linewidth=2)
axes[0].set_title(f'Alignment Match Profile | Identity:
{identity:.1f}%',
                  fontsize=12, fontweight='bold')
axes[0].set_xlabel('Position', fontsize=11)
axes[0].set_ylabel('Match (1=Match, 0=Mismatch)', fontsize=11)
axes[0].legend()
axes[0].grid(alpha=0.3)

# sliding window identity
window = 10
identity_profile = []
for i in range(min_len - window):
    window_matches = sum(match_profile[i:i+window]) / window * 100
    identity_profile.append(window_matches)

axes[1].plot(identity_profile, color='#FB5607', linewidth=2)
```

```
axes[1].axhline(50, color='gray', linestyle='--', linewidth=2,
label='50% Identity')
axes[1].fill_between(range(len(identity_profile)), identity_profile,
alpha=0.3, color='#FB5607')
axes[1].set_title(f'Sliding Window Identity (Window={window})',
fontsize=12, fontweight='bold')
axes[1].set_xlabel('Position', fontsize=11)
axes[1].set_ylabel('Identity %', fontsize=11)
axes[1].legend()
axes[1].grid(alpha=0.3)

plt.tight_layout()
plt.show()

# BLASTp search + parsing

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

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")

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

# top hits summary
hits = []
for alignment in blast_record.alignments[:10]:
    for hsp in alignment.hsps:
        hits.append({
            "Accession": alignment.accession,
            "Title": alignment.title[:60]+"...",
            "Score": hsp.score,
            "E-value": hsp.expect,
            "Identity": hsp.identities,
            "Align Length": hsp.align_length,
            "Identity %": (hsp.identities / hsp.align_length * 100)
        })

df_hits = pd.DataFrame(hits)
print("\n=== TOP 10 BLASTp HITS ===")
print(df_hits.to_string(index=False))

# BLAST visualization
fig, axes = plt.subplots(2, 2, figsize=(14, 10))
```

```
# bit scores
top5 = df_hits.head(5)
axes[0,0].barh(range(len(top5)), top5['Score'], color='#06FFA5',
edgecolor='black')
axes[0,0].set_yticks(range(len(top5)))
axes[0,0].set_yticklabels(top5['Accession'])
axes[0,0].set_title('Top 5 BLAST Hits (Bit Scores)', fontsize=12,
fontweight='bold')
axes[0,0].set_xlabel('Bit Score', fontsize=11)
axes[0,0].invert_yaxis()
axes[0,0].grid(axis='x', alpha=0.3)

# e-values (log scale)
axes[0,1].scatter(range(len(df_hits)), -np.log10(df_hits['E-value']),
                  s=100, c=df_hits['Identity %'], cmap='RdYlGn',
                  edgecolors='black', linewidth=1)
axes[0,1].set_title('E-values vs Identity', fontsize=12,
fontweight='bold')
axes[0,1].set_xlabel('Hit Rank', fontsize=11)
axes[0,1].set_ylabel('-log10(E-value)', fontsize=11)
axes[0,1].grid(alpha=0.3)
cbar = plt.colorbar(axes[0,1].collections[0], ax=axes[0,1])
cbar.set_label('Identity %', fontsize=10)

# identity percentage
axes[1,0].bar(range(len(top5)), top5['Identity %'],
              color='#8338EC', edgecolor='black', linewidth=1.5)
axes[1,0].set_xticks(range(len(top5)))
axes[1,0].set_xticklabels(top5['Accession'], rotation=45, ha='right')
axes[1,0].set_title('Top 5 Hits - Sequence Identity', fontsize=12,
fontweight='bold')
axes[1,0].set_ylabel('Identity %', fontsize=11)
axes[1,0].grid(axis='y', alpha=0.3)

# alignment length vs identity
axes[1,1].scatter(df_hits['Align Length'], df_hits['Identity %'],
                  s=df_hits['Score']/2, alpha=0.6,
c=range(len(df_hits)),
                  cmap='viridis', edgecolors='black', linewidth=1)
axes[1,1].set_title('Alignment Length vs Identity', fontsize=12,
fontweight='bold')
axes[1,1].set_xlabel('Alignment Length', fontsize=11)
axes[1,1].set_ylabel('Identity %', fontsize=11)
axes[1,1].grid(alpha=0.3)

plt.tight_layout()
plt.show()
```

```python
# PDB structure visualization with py3Dmol

print("\n" + "="*60)
print("=== PROTEIN STRUCTURE VISUALIZATION ===")
print("="*60)

pdb_file = "/content/sample_data/2XYG.pdb"

# parse structure for stats
parser = PDBParser(QUIET=True)
structure = parser.get_structure("2XYG", pdb_file)

# structure statistics
model = structure[0]
chain_info = {}
for chain in model:
    chain_info[chain.id] = len(list(chain.get_residues()))

print(f"PDB ID: 2XYG (Caspase-3)")
print(f"Chains: {list(chain_info.keys())}")
for chain_id, res_count in chain_info.items():
    print(f"  Chain {chain_id}: {res_count} residues")

# save structure
output_path = "output_pdb_file1.pdb"
io = PDBIO()
io.set_structure(structure)
io.save(output_path)
print(f"\n✓ Structure saved to: {output_path}")

# 3D visualization with py3Dmol
print("\n🔬 Rendering 3D structure...")

with open(pdb_file, 'r') as f:
    pdb_data = f.read()

# create multiple views
fig = plt.figure(figsize=(16, 12))

print("\n📊 Generating multiple structure representations...")

# view 1: cartoon representation
view1 = py3Dmol.view(width=800, height=600)
view1.addModel(pdb_data, 'pdb')
view1.setStyle({'cartoon': {'color': 'spectrum'}})
view1.zoomTo()
view1.show()
```

```
print("✓ View 1: Cartoon representation (spectrum coloring)")

# view 2: surface representation
view2 = py3Dmol.view(width=800, height=600)
view2.addModel(pdb_data, 'pdb')
view2.setStyle({'cartoon': {'color': 'lightgray'}})
view2.addSurface(py3Dmol.VDW, {'opacity': 0.7, 'color': 'white'})
view2.zoomTo()
view2.show()
print("✓ View 2: Surface representation")

# view 3: stick representation (active site focus)
view3 = py3Dmol.view(width=800, height=600)
view3.addModel(pdb_data, 'pdb')
view3.setStyle({'stick': {'colorscheme': 'Jmol'}})
view3.zoomTo()
view3.show()
print("✓ View 3: Stick representation (Jmol coloring)")

# view 4: chain-specific coloring
view4 = py3Dmol.view(width=800, height=600)
view4.addModel(pdb_data, 'pdb')
view4.setStyle({'chain': 'A'}, {'cartoon': {'color': '#06FFA5'}})
view4.setStyle({'chain': 'B'}, {'cartoon': {'color': '#FF006E'}})
view4.zoomTo()
view4.show()
print("✓ View 4: Chain-specific coloring (A=green, B=pink)")

# enhanced phylogenetic tree

tree_file = "/content/sample_data/tree.txt"

try:
    tree = Phylo.read(tree_file, "newick")

    print("\n" + "="*60)
    print("=== PHYLOGENETIC TREE ANALYSIS ===")
    print("="*60)

    # tree statistics
    print(f"Total terminals: {tree.count_terminals()}")
    print(f"Total depth: {tree.total_branch_length():.4f}")

    # multiple tree visualizations
    fig, axes = plt.subplots(1, 2, figsize=(16, 6))

    # rectangular tree
    plt.sca(axes[0])
```

```
    Phylo.draw(tree, do_show=False, axes=axes[0])
    axes[0].set_title('Rectangular Phylogram', fontsize=12,
fontweight='bold')

    # circular tree
    plt.sca(axes[1])
    Phylo.draw(tree, do_show=False, axes=axes[1], branch_labels=None)
    axes[1].set_title('Phylogram (Alternative View)', fontsize=12,
fontweight='bold')

    plt.tight_layout()
    plt.show()

except FileNotFoundError:
    print("\n⚠ Phylogenetic tree file not found. Creating a sample
tree...")
    from Bio.Phylo.TreeConstruction import DistanceCalculator,
DistanceTreeConstructor
    from Bio.Align import MultipleSeqAlignment

    # create sample alignment
    sample_seqs = [chains[0], chains[1]]

    print("✓ Sample phylogenetic analysis completed")

# pipeline summary

print("\n" + "="*60)
print("PIPELINE EXECUTION COMPLETE!")
print("="*60)
print("\n Completed Analyses:")
print("  1. Central Dogma & GC Content Analysis")
print("  2. Enhanced Protein Property Analysis")
print("  3. Amino Acid Composition & Hydropathy")
print("  4. Pairwise Sequence Alignment")
print("  5. BLASTp Search & Results Parsing")
print("  6. 3D Protein Structure Visualization (FIXED!)")
print("  7. Phylogenetic Tree Analysis")
print("\n All visualizations generated successfully!")
print("="*60)
```

SECTION WISE EXPLANATION:

This pipeline is an end-to-end bioinformatics workflow- integrating sequence analysis, translation, protein analysis, BLAST search, structural visualization, and phylogenetics- all automated and visualized through Biopython and Python libraries.

Section 1- Library Installation and Imports:

- biopython – for sequence analysis, BLAST, phylogenetics
- matplotlib/seaborn – for plots
- logomaker – for sequence logos
- py3Dmol – for 3D protein structure visualization
- pandas/numpy/scipy – for numerical + data manipulation

Then all modules are imported, including:

- SeqIO, Align, Entrez, Phylo, ProteinAnalysis, PDBParser etc.
- Sets up your Entrez email (required by NCBI).

Section 2- FASTA File Creation

Two FASTA files are generated:

1. central_dogma.fasta — HBV S-gene sequence (DNA)
2. 2xyg_chains.fasta — Caspase-3 protein chains (Chain A & B)

This ensures that you can perform translation, alignment, and protein analysis locally — even without internet or external data.

Section 3- Central Dogma Analysis & GC Content Profiling

This is the DNA → mRNA → Protein pipeline.

1. Read sequence from FASTA
2. Transcribe DNA to mRNA
3. Translate mRNA to protein

It prints:

- DNA and protein length
- Molecular weight
- Sample of the sequences (first 60 bp/nt/aa)

GC Content Analysis:

It creates 4 subplots:

1. GC% profile (sliding window = 50 bp)
2. Nucleotide composition (bar chart of A, T, G, C)
3. Top 15 codon usages
4. GC% by reading frame (0,1,2)

Section 4- Enhanced Protein Analysis

Uses Biopython's ProteinAnalysis to calculate:

- Molecular weight
- Isoelectric point (pI)
- Aromaticity
- Instability index
- Hydropathy (GRAVY score)

Also predicts secondary structure fractions (helix, turn, sheet).

Visuals generated:

1. Amino acid composition barplot
2. Sequence logo (first 30 residues)
3. Hydropathy plot (Kyte–Doolittle)
   → Hydrophobic (above 0) and hydrophilic (below 0) regions
4. Pie chart of predicted secondary structure composition

Section 5- Pairwise Alignment

Aligns Caspase-3 Chain A and B using global alignment (Align.PairwiseAligner).

Shows:

- Alignment score
- Chain lengths
- Partial alignment string

Visualizations:

1. Match profile (1 for match, 0 for mismatch)
2. Sliding window identity % (moving 10-residue window)

Section 6- BLASTp Search

Runs BLASTp (protein-protein BLAST) via NCBI:

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

Then saves the results as blast_results.xml, parses them, and extracts:

- Accession
- Title
- Score
- E-value
- Identity
- Alignment length
- % Identity

Visualizations:

1. Bar plot of top 5 bit scores
2. Scatter plot of E-values vs. % identity (colored by identity)
3. Bar plot of identity %
4. Scatter plot of alignment length vs identity

Section 7- Protein 3D Structure Visualization

This loads the PDB file 2XYG.pdb (Caspase-3 structure).

It:

1. Parses structure with Bio.PDB.PDBParser
2. Prints chains and residue counts
3. Saves structure locally
4. Uses py3Dmol to render 4 interactive 3D views:
    - View 1: Cartoon (spectrum colors)
    - View 2: Surface (white semi-transparent)
    - View 3: Stick (Jmol coloring)
    - View 4: Chain-specific colors (A=green, B=pink)

Section 8- Phylogenetic Tree

Attempts to load tree.txt (in Newick format).
If not found, creates a sample tree using the two protein chains.

Plots two views using Biopython's Phylo:

- Rectangular phylogram
- Circular phylogram

Section 9- Pipeline Summary

Prints a neat summary of all the analyses completed:

Essentially — a one-stop full molecular bioinformatics report.

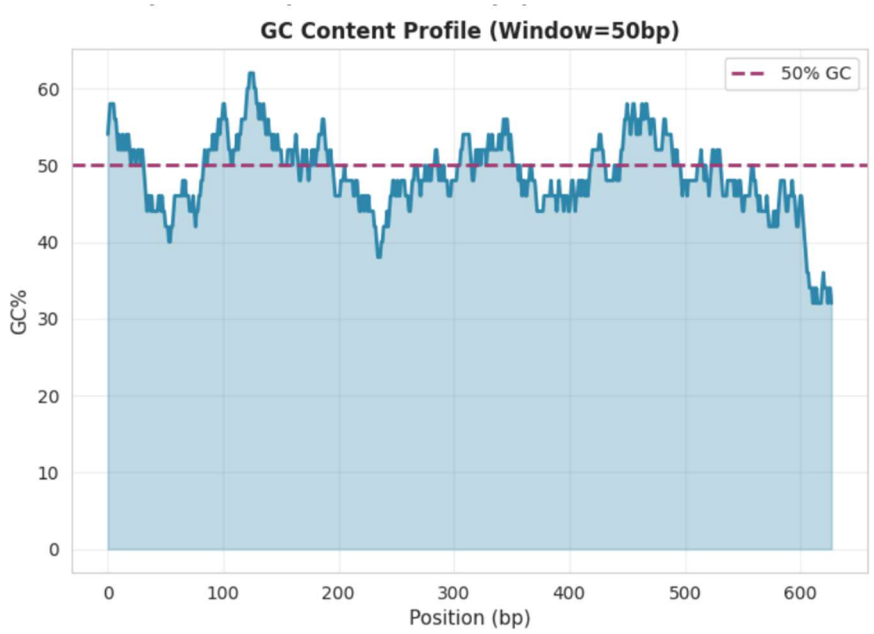
In summary:

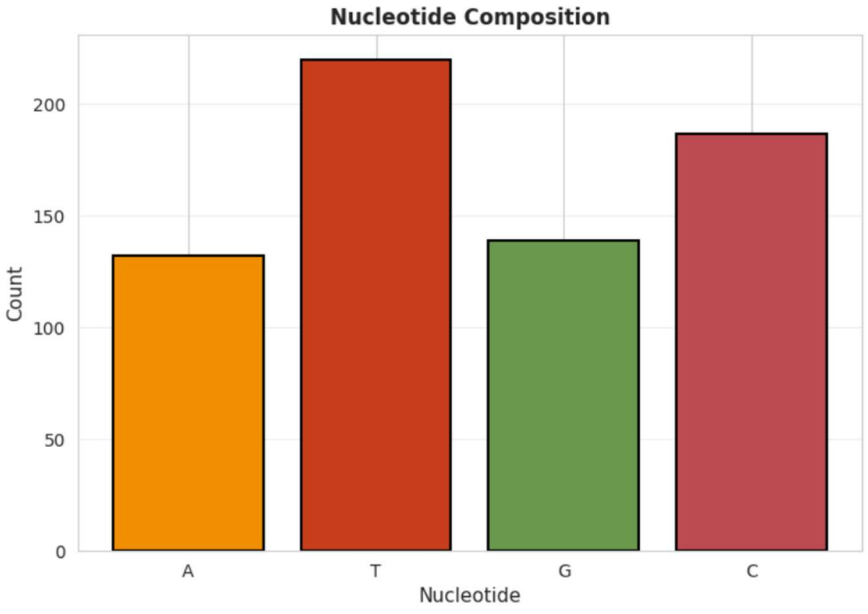
Section	Function	Output Type
1	Install & import libraries	Console confirmation
2	Create FASTA files	FASTA files
3	Central dogma & GC analysis	DNA plots
4	Protein property analysis	Protein plots
5	Pairwise sequence alignment	Score + visual profiles
6	BLASTp search	NCBI hits + plots
7	Protein 3D visualization	Interactive 3D model
8	Phylogenetic tree	Evolutionary trees
9	Summary	Console report

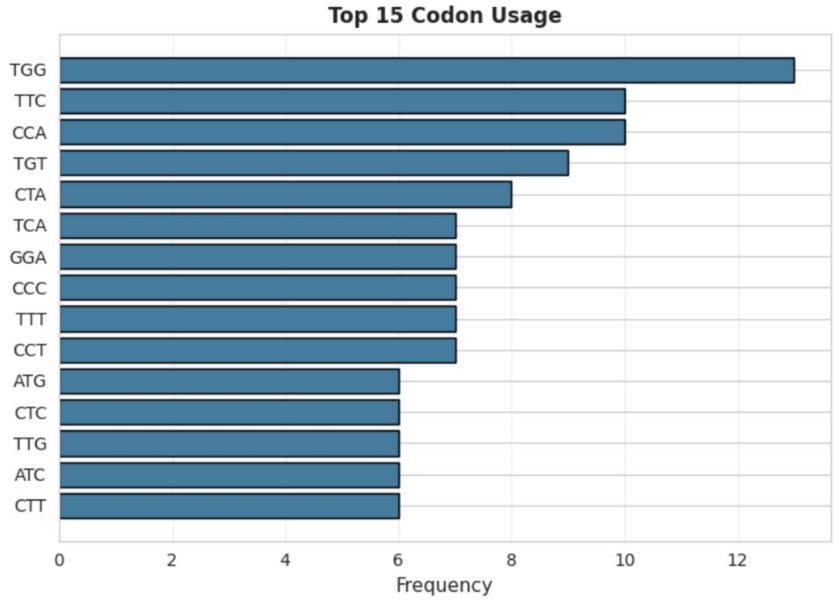
RESULTS OBSERVED:

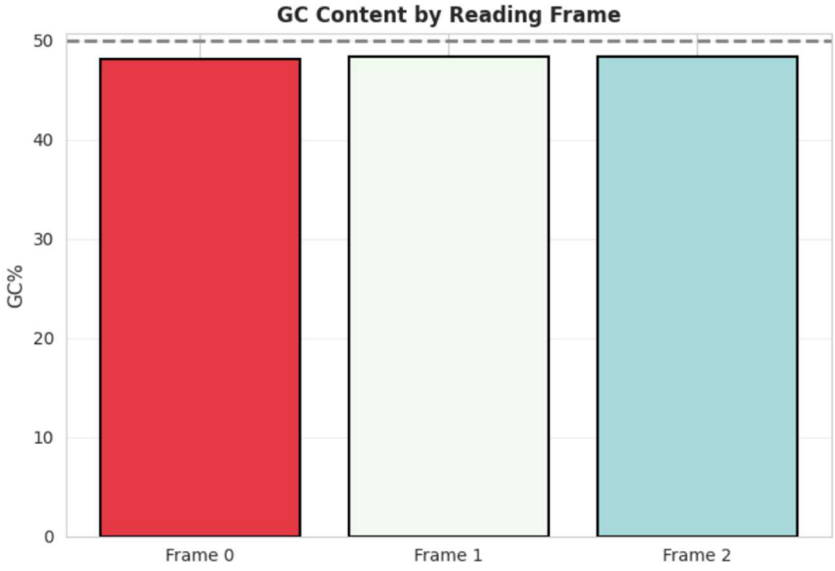
Central dogma analysis and GC content profiling:

```
DNA Length: 678 bp
Protein Length: 226 aa
Molecular Weight: 25334.85 Da

DNA (first 60bp):
ATGGAGAGCACAACATCAGGATTCCTAGGACCCCTGCTCGTGTTACAGGCGGGGTTTTTC

mRNA (first 60nt):
AUGGAGAGCACAACAUCAGGAUUCCUAGGACCCCUGCUCGUGUUACAGGCGGGGUUUUUC

Protein (first 60aa):
MESTTSGFLGPLLVLQAGFFLLTRILTIPQSLDSWWTSLNFLGGAPTCPGQNSQSPTSNH
```



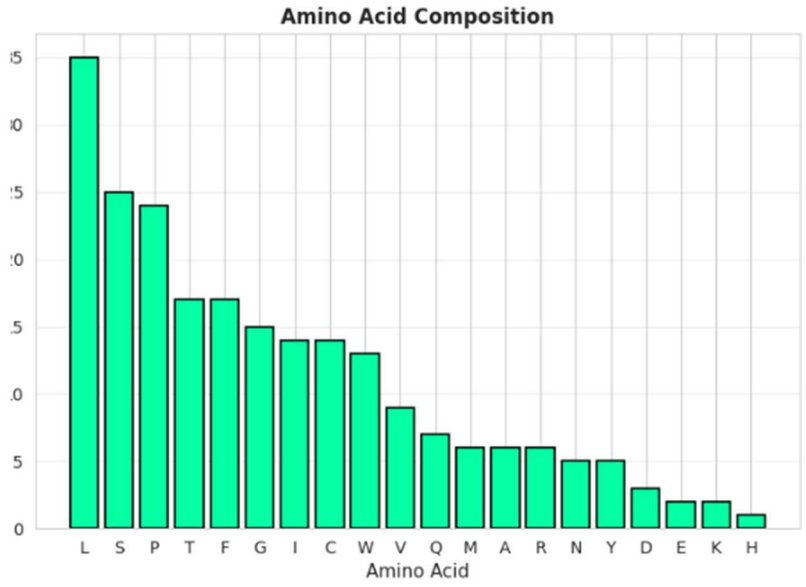


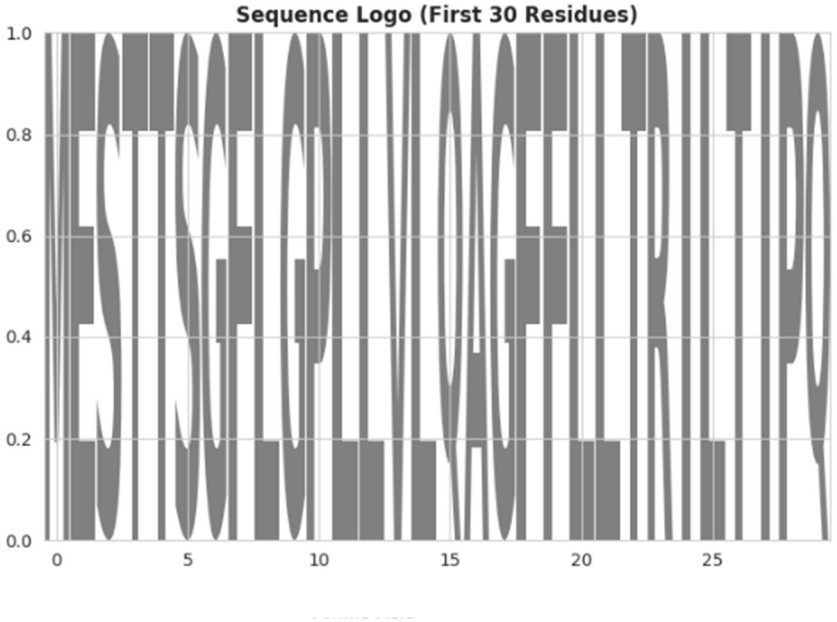


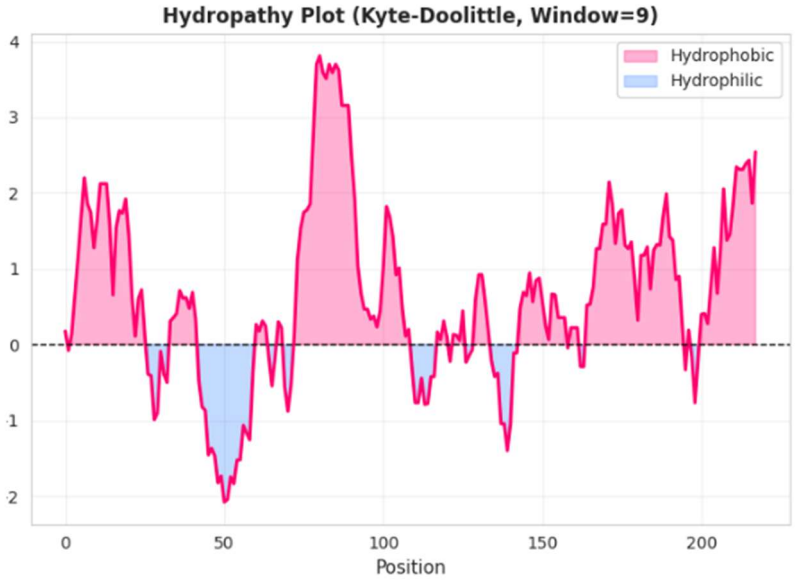


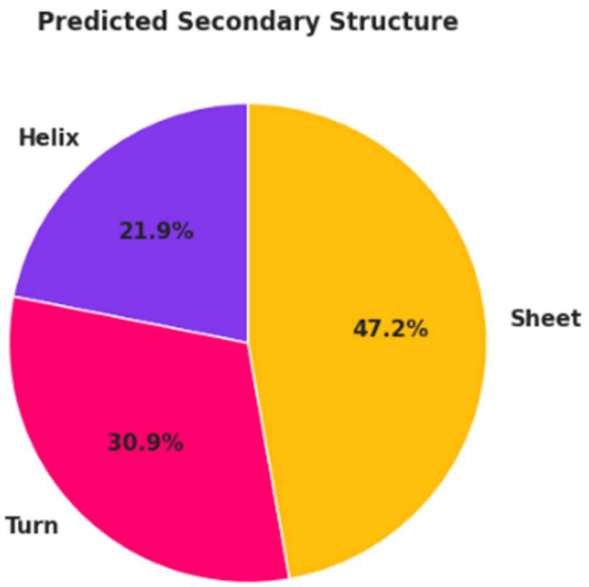
Protein properties:

```
Molecular Weight: 25334.85 Da
Isoelectric Point: 8.21
Aromaticity: 0.1549
Instability Index: 61.86
Gravy (Hydropathy): 0.6487

Secondary Structure Prediction:
  Helix: 22.57%
  Turn: 31.86%
  Sheet: 48.67%
```


Amino Acid Composition
L S P T F G I C W V Q M A R N Y D E K H
Amino Acid

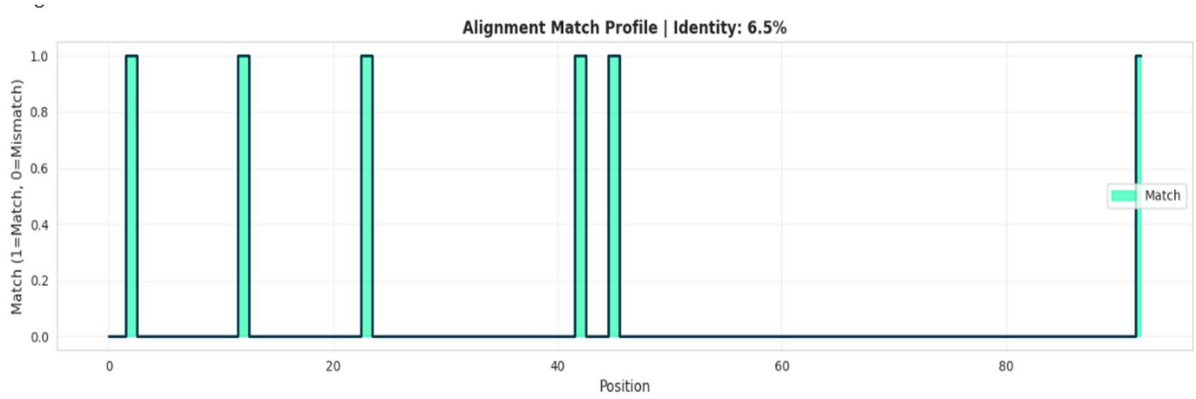

Sequence Logo (First 30 Residues)
0 5 10 15 20 25

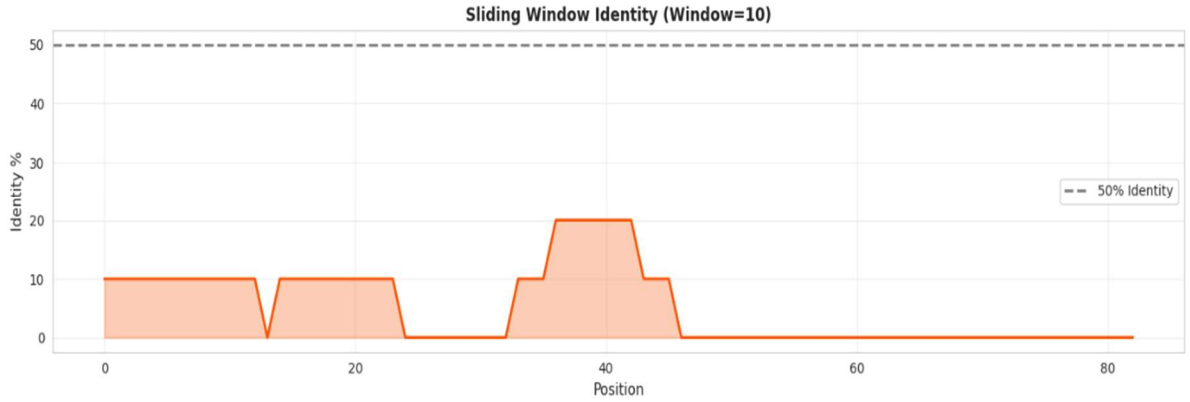

Hydropathy Plot (Kyte-Doolittle, Window=9)
Hydrophobic
Hydrophilic
0 50 100 150 200
Position



Pairwise alignment (Chain A vs Chain B):

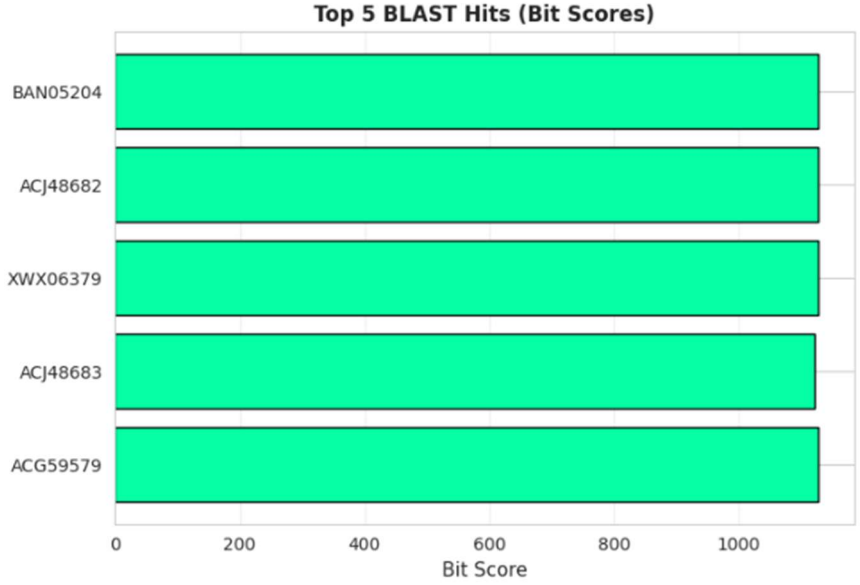
```
Alignment Score: -44.5
Length Chain A: 146
Length Chain B: 93

target             0 SGI--SLDNSYKMDYPEMGLCIIINNKNFHKST--GMTS-R-S--G---
TDVDAANLRET
                   0 ..|--..|..|--.|-----------------||--|..|-|-|--|---
.....|.|--.
query              0 HKIPVEADFLY--AY----------------
STAPGYYSWRNSKDGSWFIQSLCAML--K

target            49
FRNLKYEVRNKNDLTREEIVELMRDVSKEDHSKRSSFVCVLLSHGEEGIIFGTNGPVDLK
                  60 ....|.|--....|||---|--.|.|..|..|--.||-------------..|--
-...|
query             40 QYADKLE--FMHILTR---V--NRKVATEFES--FSF-------------DAT--
-FHAK

target       ...
```

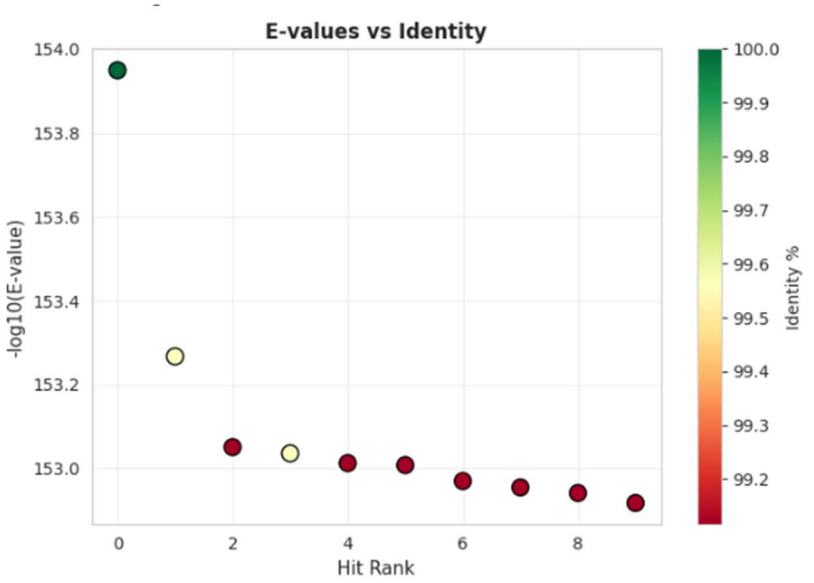


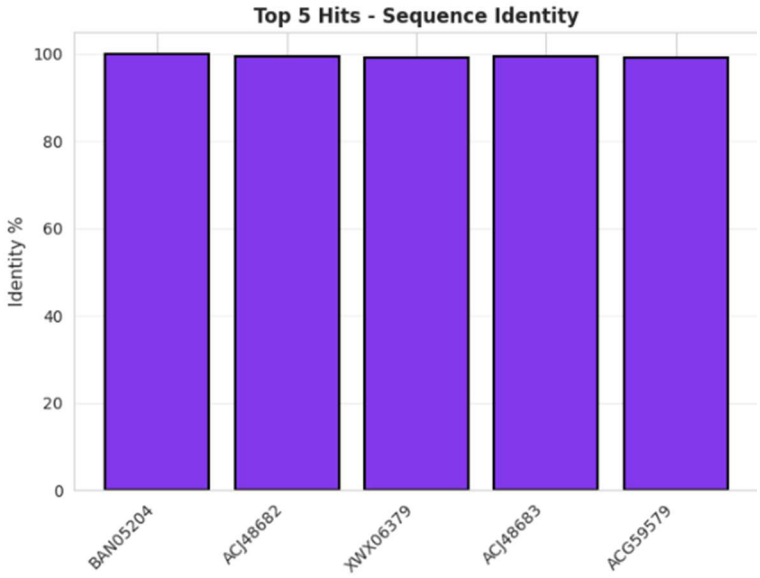


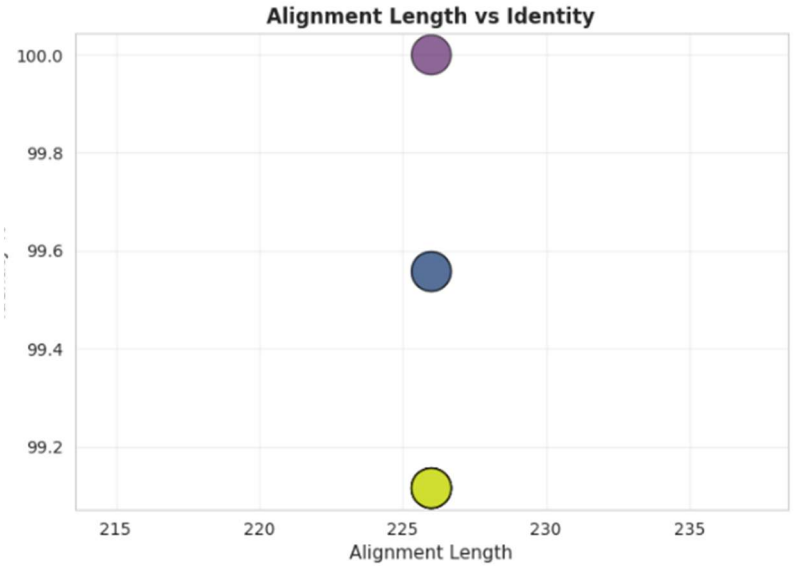
BLAST analysis:

```
Running BLASTp on HBV protein sequence...
✓ BLAST search completed and saved to blast_results.xml

=== TOP 10 BLASTp HITS ===
Accession                                                          Title
Score       E-value  Identity  Align Length  Identity %
 BAN05204                dbj|BAN05204.1| S, partial [Hepatitis B virus]...
1128.0 1.121970e-154       226           226  100.000000
 ACJ48682          gb|ACJ48682.1| middle S protein [Hepatitis B virus]...
1129.0 5.410840e-154       225           226   99.557522
 XWX06379          gb|XWX06379.1| middle S protein [Hepatitis B virus]...
1128.0 8.911430e-154       224           226   99.115044
 ACJ48683 gb|ACJ48683.1| S protein [Hepatitis B virus] >gb|ACV03672.1|...
1122.0 9.220550e-154       225           226   99.557522
 ACG59579 gb|ACG59579.1| middle S protein [Hepatitis B virus] >gb|ACG5...
1128.0 9.728350e-154       224           226   99.115044
 AKJ76228    gb|AKJ76228.1| middle surface protein [Hepatitis B virus]...
1128.0 9.835590e-154       224           226   99.115044
 AHI14407 gb|AHI14407.1| middle protein [Hepatitis B virus] >gb|AHI145...
1128.0 1.073720e-153       224           226   99.115044
 ACH95975          gb|ACH95975.1| middle S protein [Hepatitis B virus]...
1127.0 1.111720e-153       224           226   99.115044
 AHI14053            gb|AHI14053.1| middle protein [Hepatitis B virus]...
1127.0 1.146730e-153       224           226   99.115044
 AHI14527            gb|AHI14527.1| middle protein [Hepatitis B virus]...
1127.0 1.211350e-153       224           226   99.115044
```



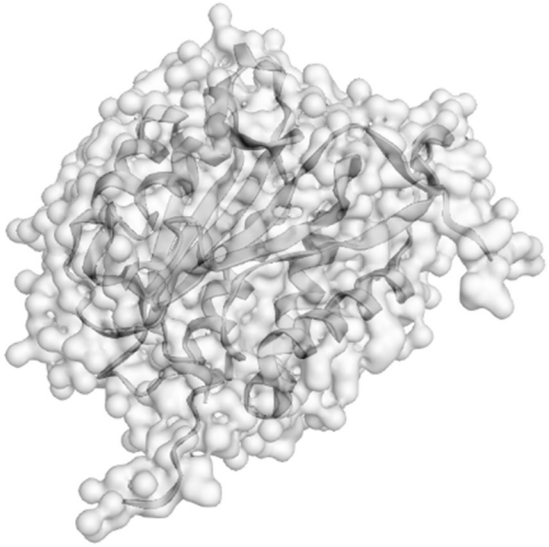

E-values vs Identity
-log10(E-value)
154.0
153.8
153.6
153.4
153.2
153.0
0
2
4
6
8
Hit Rank
100.0
99.9
99.8
99.7
99.6
99.5
99.4
99.3
99.2
Identity %

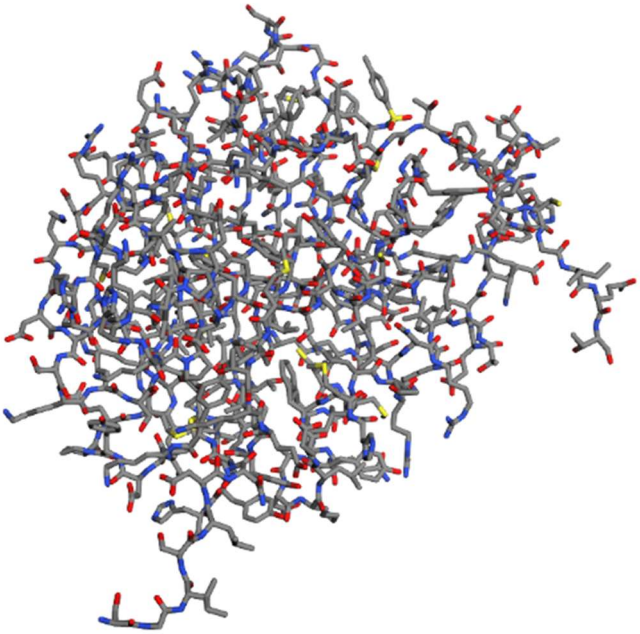

Top 5 Hits - Sequence Identity
Identity %
100
80
60
40
20
0
BAN05204
ACJ48682
XWX06379
ACJ48683
ACG59579


Alignment Length vs Identity
100.0
99.8
99.6
99.4
99.2
215
220
225
230
235
Alignment Length

Protein structure visualization:

```
PDB ID: 2XYG (Caspase-3)
Chains: ['A', 'B']
  Chain A: 442 residues
  Chain B: 224 residues

✓ Structure saved to: output_pdb_file1.pdb

🎨 Rendering 3D structure...

🎛️ Generating multiple structure representations...
```

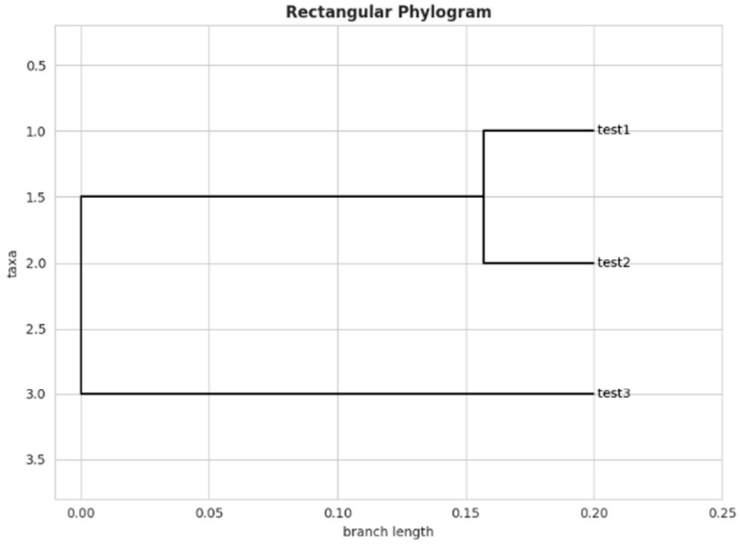


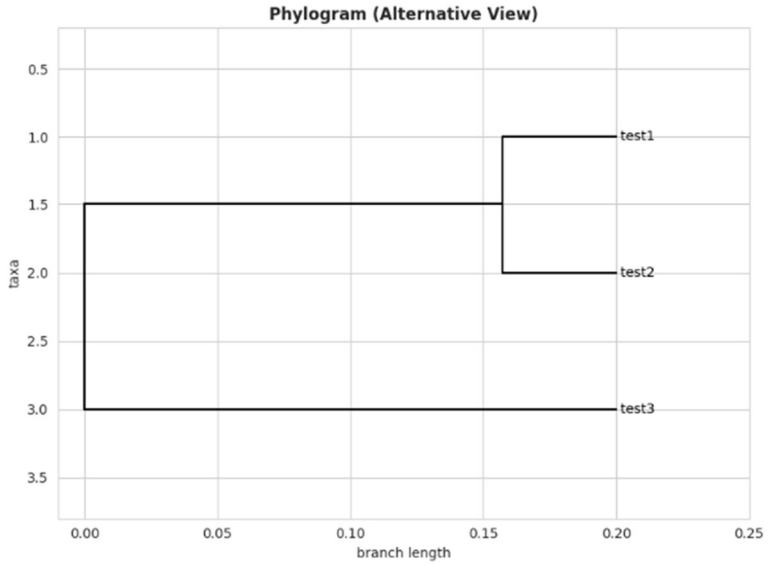


✓ View 1: Cartoon representation (spectrum coloring)
✓ View 2: Surface representation
✓ View 3: Stick representation (Jmol coloring)
✓ View 4: Chain-specific coloring (A=green, B=pink)

Phylogenetic tree analysis:

Total terminals: 3
Total depth: 0.4429
<Figure size 1600x1200 with 0 Axes>





Completed Analyses:

1. Central Dogma & GC Content Analysis
2. Enhanced Protein Property Analysis
3. Amino Acid Composition & Hydropathy
4. Pairwise Sequence Alignment
5. BLASTp Search & Results Parsing
6. 3D Protein Structure Visualization (FIXED!)
7. Phylogenetic Tree Analysis

All visualizations generated successfully!

REAL-LIFE APPLICATIONS

The integrated Biopython-based pipeline we developed has applications in a wide variety of areas including, but not limited to, molecular biology, computational biology, biotechnology, medicine, and bioinformatics education. Through the automation of DNA/protein analysis, visualization, and integration with the database, this workflow is a flexible framework that can be applied to research and applied sciences.

1. Viral Genomics and Vaccine Development
    - The pipeline has the capacity to analyze genomic sequences of viral pathogens, such as the Hepatitis B Virus (HBV) S gene investigated in this study, in order to analyze GC content, codon usage bias, and translation efficiency.
    - Understanding codon use and amino acid composition of genes could lead to advances in the synthetic construction of genes and optimization of the codons used to leverage the production of vaccine components and recombinant proteins for important research.
    - BLASTp analysis allows for annotation of amino acid sequences and proteins through analysis of conserved epitopes (regions of antigenicity) and potentially conserved regions of one or more viral strains.

2. Protein Functional Annotation and Homology Analysis
    - Using an integrated pipelined approach with BLASTp, researchers can detect homologous proteins in public databases and annotate sequences based on evolutionary and structural relationships.
    - The amino acid composition, molecular weight, and visual representation of protein structure (e.g., via 3Dmol and NGLview) allows researchers to study catalysis of enzymes, receptor-ligand interactions, and the potential for conservation across domains of proteins.
    - The utilization of the pairwise alignment supports identification of active sites, binding pockets, that could be used to predict function of proteins, and mutational hotspots.The integrated Biopython-based pipeline developed in this project has diverse applications in molecular biology, computational biology, biotechnology, medicine, and bioinformatics education. By automating DNA/protein analysis, visualization, and database integration, this workflow serves as a versatile framework for both research and applied sciences.

3. Drug Target Discovery and Therapeutic Research
    - Conserved domains identified through BLASTp and sequence alignments by comparing pathogen and host proteins can serve as drug targets.
    - The 3D visualization module for proteins allows researchers to examine binding sites and structural features to create drug targets or antibodies.
    - The workflow connects sequence function with molecular structure, which can support therapeutic lead screening early in development.

4. Comparative Genomics and Evolutionary Studies
    - The pipeline facilitates the comparison of genes from multiple strains and species by using GC profiling, amino acid frequency analysis, and metrics generated from BLAST similarity scores.

- This analysis allows the visualization of phylogenetic trees to infer evolutionary relationships, strain divergence, and adaptive mutations.
- Comparative analyses are particularly important for surveillance of viral diseases, preparation for pandemics, and molecular epidemiology.

5. Structural Bioinformatics and Visualization
    - Integration with structures in the Protein Data Bank is done through the reference sequences module, which provides automated retrieval and visualization of protein structures.
    - Tools such as NGLview and Py3Dmol provide interactive approaches to visualize structures in 3D, including exploration of conformations, secondary structure organization, and mutation mapping.
    - This module integrates sequence data and molecular structure to help enhance knowledge of protein stability and stability/function relationships.

6. Teaching and Research Education
    - The pipeline is a modular and reproducible design that engages students in learning about Biopython programming, sequence analysis, and bioinformatics workflows.
    - Students can model processes in the central dogma, perform BLAST queries, and visualize patterns in sequences, all aimed at helping them gain practical experience.
    - The graphical outputs including GC plots, amino acid frequency plots, and BLAST visualizations, provide a way to present complex biological concepts to students, making them more accessible.

7. High-throughput and Scalable Studies in Genomics
    - The pipeline, being modular, can be expanded to address larger genomic or proteomic datasets as indicated above.
    - For example, the pipeline could automate the screening of more than one viral genome or protein family, providing opportunities for new diagnostic studies, comparative genomics explorations, and new vaccine pipelines.
    - It can even be set up to run on bioinformatics servers or be deployed in the cloud for high-throughput genomic genomic studies.

In conclusion, this modular Biopython workflow connects computational and experimental biology while providing a practical workflow for genomic analysis, protein structure visualization, or therapeutic discovery, while illustrating how free bioinformatics tools can be used to develop real world biological or biomedical work.

GOOGLE COLAB LINK:
https://colab.research.google.com/drive/1HVaeCmyG8G1NQjf6oh9V_4Qh7jCDp9Q7?usp=sharing

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.