We'll begin with the core functionality — reading sequences, aligning them, and generating results. This code will later be split into separate scripts in the **src/** folder (as shown in the project structure).

**Step-by-Step Python Code**

**1. Reading the Input Files**

We'll start by reading the input FASTA files (dog\_breeds.fa and mystery.fa) using **BioPython's SeqIO** module. This will allow us to load the sequences into Python and start processing them.

**File: src/sequence\_matcher.py**

from Bio import SeqIO

def read\_sequences(file\_path):

"""Reads sequences from a FASTA file."""

try:

sequences = list(SeqIO.parse(file\_path, "fasta"))

return sequences

except FileNotFoundError:

print(f"Error: The file {file\_path} was not found.")

return []

except Exception as e:

print(f"Error while reading the file {file\_path}: {e}")

return []

def main():

# Define file paths for data files

dog\_breeds\_file = "data/dog\_breeds.fa"

mystery\_file = "data/mystery.fa"

# Read sequences from the files

dog\_breeds\_sequences = read\_sequences(dog\_breeds\_file)

mystery\_sequences = read\_sequences(mystery\_file)

if dog\_breeds\_sequences and mystery\_sequences:

print("Files successfully read!")

else:

print("Error reading files.")

return

# Further processing will be done here (alignment and matching)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**2. Sequence Alignment (Using pairwise2)**

Next, we'll implement the sequence alignment using **pairwise2**, which is available in **BioPython**. We will align the mystery sequence (mystery.fa) with each sequence in the database (dog\_breeds.fa) and calculate an alignment score.

**File: src/align\_sequences.py**

from Bio import pairwise2

def align\_sequences(seq1, seq2):

"""Align two sequences using global alignment and return the best alignment."""

alignments = pairwise2.align.globalxx(seq1, seq2) # Perform global alignment

if alignments:

return alignments[0] # Return the best alignment (highest score)

return None

def find\_best\_match(test\_sequence, database\_sequences):

"""Find the best matching sequence from the database."""

best\_match = None

best\_score = -float('inf') # Initialize with the lowest possible score

for db\_sequence in database\_sequences:

alignment = align\_sequences(test\_sequence.seq, db\_sequence.seq)

if alignment: # Check if alignment exists

score = alignment[2] # Alignment score is in index 2

if score > best\_score: # Keep track of the best match

best\_score = score

best\_match = db\_sequence

return best\_match, best\_score

**3. Output the Results**

We’ll store the best match sequence and its alignment score in a **text file** called classification\_results.txt in the **results/** folder.

**File: src/sequence\_matcher.py (continued)**

from align\_sequences import find\_best\_match

from Bio import SeqIO

def save\_results(best\_match, best\_score):

"""Save the best match and score to a results file."""

with open("results/classification\_results.txt", "w") as result\_file:

result\_file.write(f"Best Match: {best\_match.id}\n")

result\_file.write(f"Alignment Score: {best\_score}\n")

def main():

# Define file paths

dog\_breeds\_file = "data/dog\_breeds.fa"

mystery\_file = "data/mystery.fa"

# Read sequences

dog\_breeds\_sequences = read\_sequences(dog\_breeds\_file)

mystery\_sequences = read\_sequences(mystery\_file)

if not dog\_breeds\_sequences or not mystery\_sequences:

print("Error: Missing or empty sequences. Exiting.")

return

# Assuming we have only one mystery sequence

test\_sequence = mystery\_sequences[0]

# Find the best match from the dog breed database

best\_match, best\_score = find\_best\_match(test\_sequence, dog\_breeds\_sequences)

# Save the result to a file

if best\_match:

print(f"Best match: {best\_match.id} with a score of {best\_score}")

save\_results(best\_match, best\_score)

else:

print("No close match found.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**4. Project Testing (Testing File)**

We’ll write some basic tests for reading the sequences, aligning them, and saving the output. These tests will go in the **tests/** folder.

**File: tests/test\_sequence\_matching.py**

import unittest

from src.sequence\_matcher import read\_sequences

from Bio import SeqIO

class TestSequenceMatcher(unittest.TestCase):

def test\_read\_sequences\_valid(self):

# Test if the FASTA file is read correctly

sequences = read\_sequences("data/dog\_breeds.fa")

self.assertGreater(len(sequences), 0) # Ensure sequences were loaded

def test\_read\_sequences\_invalid(self):

# Test if the function handles invalid files gracefully

sequences = read\_sequences("data/nonexistent\_file.fa")

self.assertEqual(sequences, []) # No sequences should be read

def test\_find\_best\_match(self):

# Test that the correct best match is found

dog\_breeds\_sequences = list(SeqIO.parse("data/dog\_breeds.fa", "fasta"))

mystery\_sequences = list(SeqIO.parse("data/mystery.fa", "fasta"))

best\_match, best\_score = find\_best\_match(mystery\_sequences[0], dog\_breeds\_sequences)

self.assertIsNotNone(best\_match) # Ensure a match is found

self.assertGreater(best\_score, 0) # Ensure score is positive (indicating a match)

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

**5. Explanation of the Python Code**

**1. sequence\_matcher.py**

* **read\_sequences**: Reads sequences from a FASTA file and returns them as a list.
* **save\_results**: Saves the closest match and alignment score to a text file in the results/ folder.
* **main**: The main function that coordinates the process of reading files, finding the best match, and saving the results.

**2. align\_sequences.py**

* **align\_sequences**: Uses pairwise2.align.globalxx() to align two sequences globally. This function compares two sequences and returns the best alignment.
* **find\_best\_match**: Loops through the sequences in the database and finds the one with the highest alignment score compared to the test sequence.

**3. test\_sequence\_matching.py**

* **Unit tests**: Tests for reading sequences, matching them, and handling invalid files. This ensures the integrity of the input/output logic.

**Next Steps**

* **Run the code**: Start with reading the sequences and performing alignment on a few test cases.
* **Expand the functionality**: As you move forward, consider implementing the stretch goals like probability calculations or phylogenetic tree reconstruction.