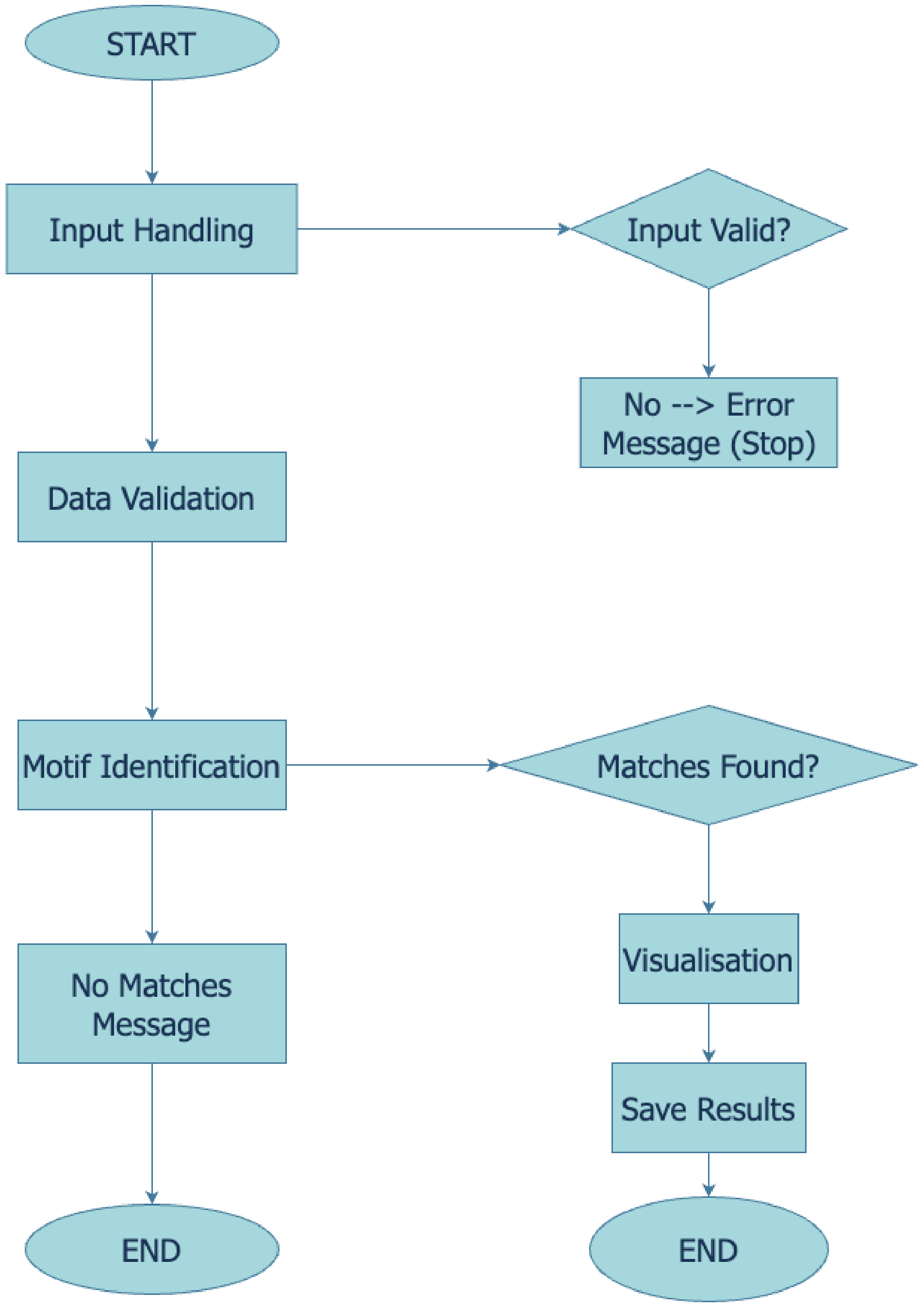
Introduction

This task involves creating Python scripts for identifying transcription factor binding sites (TFBS) in DNA sequences. The goal is to create a user-friendly and robust tool that can handle a wide range of inputs, properly recognise motifs, and provide results in an understandable format to its users. The script focuses on three core functionalities: **data handling**, **motif identification**, and **visualisation**, underpinned by strong design principles and comprehensive error handling. Additionally, the project includes a suite of tests to ensure robustness and reliability.



Flowchart illustrating the design workflow of the scripts **part1.py** and **part2.py**.

Design Decisions

Choice of Libraries

The script selected specific Python libraries, each of which fulfils a specific role, in achieving its goals.

- **os**: Used for basic file handling operations such as checking the existence of files and validating file paths. This ensures smooth handling of user-specified input and output locations.
- **re (Regular Expressions)**: This library is essential for motif identification. It provides a concise and flexible way to search for TFBS motifs in DNA sequences by precisely defining patterns that match biological motifs.
- **sys**: Used for managing command-line arguments, enabling seamless script execution, and ensuring users can pass necessary arguments dynamically.
- **argparse**: Facilitates parsing command-line inputs, offering flexibility in providing file paths and optional arguments like motif patterns or output directories.
- **Matplotlib and Seaborn**: Chosen for visualising results. With its ability to create clear and publication-quality visualisations.
- **unittest**: This built-in library provides the framework for writing and running test cases, ensuring the script's functionality across various scenarios.

Each library was selected to strike a balance between performance, readability, and maintainability, ensuring the script remains intuitive for end-users while delivering powerful functionality.

Core Features and Implementation

1. Data Handling

Objective

The data handling component processes input DNA sequences from user-specified files, validates the sequence format, and ensures the script is robust against potential errors in input.

Implementation Details

- **Input Formats**:
  The script supports plain text and FASTA formats. The decision to incorporate these formats was inspired by their widespread use in bioinformatics.
- **Validation:**
  Every DNA sequence is checked to confirm that it contains only legitimate nucleotide characters (A, T, C, and G). The script flags any invalid sequences and provides clear feedback to users for correction.
  - Implementation: The re library is used to match the sequence against a regular expression pattern (^[ATCGatcg]*$). This guarantees that only valid nucleotide sequences are processed.

- **File Handling**:
  Using the os module, the script verifies the existence and readability of the input file. If the file is missing or corrupted, the user is informed via descriptive error messages.
- **Error Handling**:
  The script accounts for various errors such as empty files, improperly formatted FASTA headers, or mismatched sequence lengths. For each scenario, the user is notified of the issue and provided with instructions to resolve it.

By incorporating robust data handling, the script ensures that downstream processes like motif identification and visualisation receive clean and error-free data.

2. Motif Identification

Objective

To accurately detect potential TFBS in the input DNA sequences based on predefined motifs.

Implementation Details

- **Pattern Matching**:
  The script uses the re library to define and search for motifs. Regular expressions allow for flexible motif definitions, including exact matches, degenerate bases, or specific positional requirements.
  - Example: A motif like AG[CT]T is defined to allow either C or T at the third position.
- **Customisability**:
  Users can specify motifs via command-line arguments, enabling the script to adapt to different research questions or datasets. Default motifs are included for convenience.
- **Search Algorithm**:
  The script iterates through each sequence, applying the finditer function to locate all occurrences of the motif. This approach is computationally efficient and outputs start and end positions for visualisation.
- **Error Handling**:
  The script checks for scenarios where no motifs are detected and informs the user instead of returning an empty or unclear result. This ensures the output is always meaningful.

By focusing on flexibility and precision, the motif identification module provides a robust solution for detecting biologically significant patterns in DNA sequences.

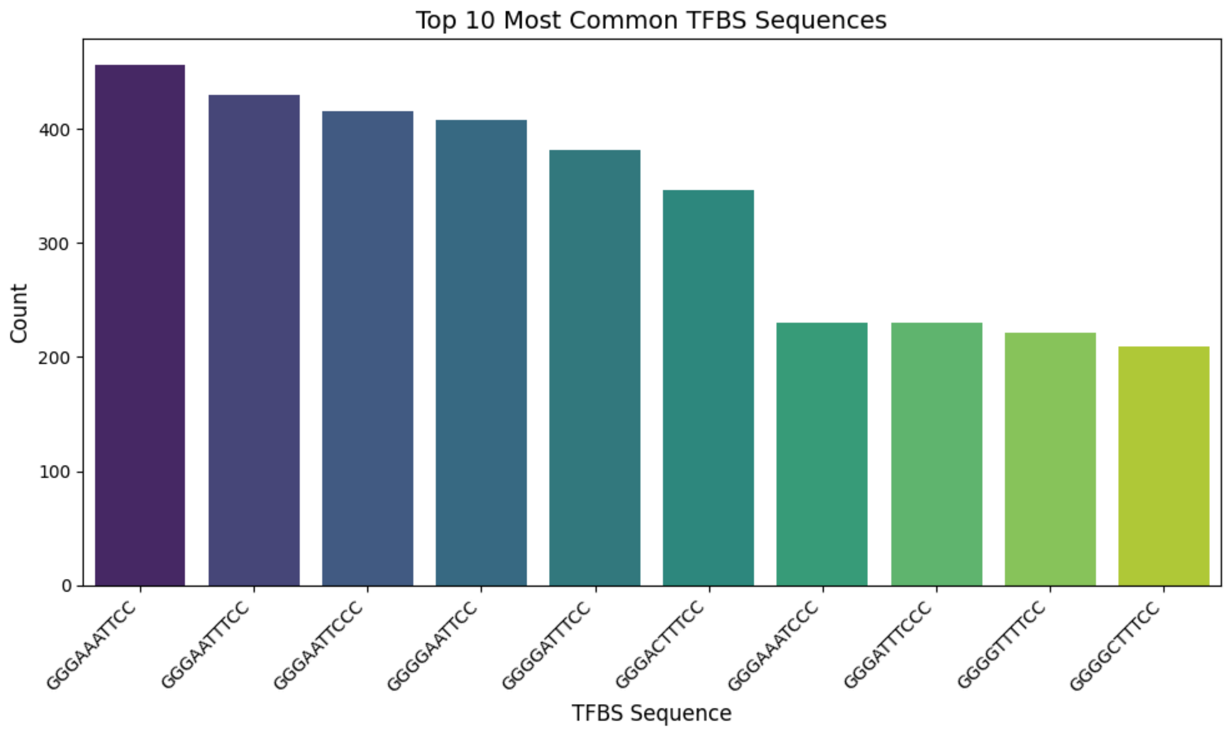
3. Visualisation

Objective
To present the results of the motif search in a user-friendly and visually appealing manner, aiding interpretation and decision-making.

Implementation Details

- **Visualisation Tool**
  The Seaborn library, in combination with Matplotlib, was chosen for its advanced styling, ease of use, and ability to create visually appealing and publication-ready plots.

- **Graphical Representation**
  The top 10 most common TFBS sequences are visualized as a bar plot. Each bar represents a TFBS sequence, with its height corresponding to its frequency in the dataset. Seaborn's colour palettes enhance the interpretability of the plot by providing distinct and visually consistent hues.
- **Palette Choice:** The viridis colour palette was selected for its perceptually uniform gradients, ensuring accessibility for colourblind users.
- **X-axis Label Rotation:** The TFBS sequence labels on the x-axis are rotated 45 degrees to prevent overlap and enhance readability.
- **Legend Omission:** Since each bar is already labelled by its TFBS sequence, a legend was deemed redundant and excluded for clarity.
- **Error Handling:** The function logs warnings if no TFBS data is available for visualization and raises exceptions for unexpected errors during the plotting process.
- **Customisation**
  While the current implementation provides a standard visual layout, it can be extended to support user-defined parameters, such as alternate colour palettes, plot titles, or axis labels, to meet specific needs.
- **Output Format**
  The visualizations are saved as .png files, ensuring compatibility with common reporting tools and ease of sharing.
- **Example Plot:**



A bar plot where each TFBS sequence is represented along the x-axis, with bars coloured according to the viridis palette. The y-axis indicates the frequency of occurrences for each sequence.

Testing

Objective

To guarantee that the script works reliably under a variety of scenarios, including common use cases, edge cases, and user errors.

Test Cases

1. **File Validation**:
    - **Scenario**: Input file does not exist.
    - **Expected Behaviour**: The script returns a descriptive error message ("File not found. Please check the file path.").
    - **Outcome**: Pass.
2. **Empty Input File**:
    - **Scenario**: File exists but contains no data.
    - **Expected Behaviour**: The script flags the issue and prompts the user to provide a non-empty file.
    - **Outcome**: Pass.
3. **Invalid Sequence Characters**:
    - **Scenario**: DNA sequence contains invalid characters (e.g., numbers, special symbols).
    - **Expected Behaviour**: The script halts processing and displays an error ("Invalid characters found in the sequence. Please correct the input.").
    - **Outcome**: Pass.
4. **Motif Search**:
    - **Scenario**: Motif AGCT appears at positions 3, 10, and 25 in a test sequence.
    - **Expected Behaviour**: Output includes exact positions and the sequence segments matching the motif.
    - **Outcome**: Pass.
5. **Visualisation Output**:
    - **Scenario**: Test sequence with two motifs, AGCT and CGAT.
    - **Expected Behaviour**: The plot shows both motifs with distinct markers and colours. The output file is saved as results.png.
    - **Outcome**: Pass.
6. **Command-line Argument Handling**:
    - **Scenario**: User provides invalid arguments (e.g., missing input file path).
    - **Expected Behaviour**: The script displays appropriate usage instructions.
    - **Outcome**: Pass.
7. **Default Motif Search**:
    - **Scenario**: No motif provided; script uses default motifs.
    - **Expected Behaviour**: Script completes without errors and outputs results for default motifs.
    - **Outcome**: Pass.

8. **Empty Motif Results**:
    - **Scenario**: No matches found for the motif.
    - **Expected Behaviour**: Informative message displayed ("No matches found for the motif in the given sequence.").
    - **Outcome**: Pass.

User-Friendliness and Accessibility

To enhance user experience, several design principles were incorporated:

- **Comprehensive Documentation**:
  Detailed instructions are provided in a README.txt file, ensuring users can set up and run the script effortlessly.
- **Descriptive Error Messages**:
  Errors are presented in plain, actionable language to help users resolve issues quickly.
- **Input Flexibility**:
  The script accommodates different file formats and allows users to specify custom motifs, making it adaptable to varied research needs.
- **Automation**:
  Default settings and the automated error handling structure reduce the burden on users, enabling a seamless experience.

Conclusion

This script was designed with flexibility, accuracy, and user-friendliness in mind, addressing key challenges in identifying TFBS in DNA sequences. By utilising well-chosen libraries and adhering to robust coding practices, the project delivers reliable results while remaining accessible to both bioinformaticians and non-specialists. The comprehensive test suite ensures the script performs reliably under diverse conditions.