Explanation of Data Structures, Edge Case Handling, and Avoiding Overcounting.

Data Structures Used to Store K-mers and Their Context.

To efficiently store and analyze k-mers and their contexts, two primary data structures were used: k-mer frequency dictionary and context dictionary. The k-mer frequency dictionary stores the total frequency of each k-mer across all input sequences. For example, if the sequence is "ATGTCTGTCTGAA" and “k=2”, the k-mer "TG" appears three times, so its frequency is stored as 3. The context dictionary is a nested structure that associates each k-mer with the characters that immediately follow it and their respective frequencies. For instance, the k-mer "TG" is followed by "T" twice and "A" once, so this information is stored as (T": 2, "A": 1). These data structures were chosen because dictionaries provide efficient storage and retrieval, making them ideal for counting and analyzing k-mers and their contexts.

Handling Edge Cases.

Edge cases, such as the first and last k-mers in a sequence, were carefully considered to ensure accurate results. The first k-mer in a sequence does not have any preceding context, but it still has a subsequent character, so it is included in both the k-mer frequency dictionary and the context dictionary. The last k-mer in a sequence, however, may not have a subsequent character. To handle this, the script allows the subsequent character to be “None” for the last k-mer, ensuring that it is still counted in the analysis. Additionally, sequences shorter than the specified value of “k” are skipped entirely, as they cannot produce valid k-mers. Invalid inputs, such as improperly formatted files or non-integer values for “k”, are also handled gracefully by notifying the user and exiting the script.

Avoiding Overcounting or Missing Context.

The code avoids overcounting by processing each k-mer and its subsequent character exactly once per occurrence in the input sequences. This ensures that the frequencies stored in the dictionaries are accurate and not inflated due to repeated processing. To avoid missing context, the script includes all k-mers, even if they do not have a subsequent character. For example, the last k-mer in a sequence is still counted, with its subsequent

character set to “None”. Furthermore, the modular design of the script, with separate functions for parsing, generating k-mers, and updating counts, prevents duplication of logic. Comprehensive testing using `pytest` ensures that the script behaves correctly under both typical and edge cases, providing accuracy in the results.