centralMolBio

The ‘CentralMolBio’ package, is a collection of useful functions for analysing biological data. including reading, subtitute and translating sequences.

Installation

The package can be install either of these methods: - devtools::install_github(“Superstinse99/Group_10_package”) - install.packages(“Group_10_package”)

~~~
# install.packages("devtools")
devtools::install_github("Superstinse99/group_10_package")




```r
library(centralMolBio)
~~~

Function 1: The first function in the package is generating a random sequence of the bases in the desired length

```
generateRandomDNASequence <- function(sequenceLength){
  randomNucleotides <- sample(c("A", "T", "G", "C"),
                              size = sequenceLength, replace = TRUE)
  randomDNASequence <- paste0(randomNucleotides, collapse = "")
  return(randomDNASequence)
}
```

Function 2: The second function in the package is a translation, where T is replaced with U, therefore transcribing DNA to RNA

```
substitueDNARNA <- function(DNA_sequence){
  RNA_sequence <- gsub("T", "U", DNA_sequence)
  return(RNA_sequence)
}
```

Function 3: The third function in the package is taking in the DNA sequence, indicating start of the sequence at one, and then reading it in codons, also known as the reading frame of the sequence.

```
settingReadingFrameCodons <- function(dnaSequence, start = 1){
  sequenceLength <- nchar(dnaSequence)
  codons <- substring(dnaSequence,
                      first = seq(from = start, to = sequenceLength - 3 + 1, by = 3),
                      last = seq(from = 3 + start - 1, to = sequenceLength, by = 3))
  return(codons)
}
```

Function 4: The fourth function in the package is a translation of the RNA, generating the translated amino acids of the sequence.

```
generateAAFromCodons <- function(codons){
  AA <- paste0(codon_table[codons], collapse = "")
  return(AA)
}
```

Function 5: The fifth function in the package is plotting the frequency of each amino acid observed in the sequence.

```r
generateAminoAcidFrequencyPlot <- function(Sequence){
  amino_acids <- Sequence |>
    stringr::str_split(pattern = stringr::boundary("character"), simplify = TRUE) |>
    as.character() |>
    unique()

  amino_acid_counts <- sapply(amino_acids, function(amino_acid) stringr::str_count(string = Sequence, pa
    as.data.frame()

  colnames(amino_acid_counts) <- c("Counts")
  amino_acid_counts[["AminoAcid"]] <- rownames(amino_acid_counts)

  frequency_plot <- amino_acid_counts |>
    ggplot2::ggplot(ggplot2::aes(x = AminoAcid, y = Counts, fill = AminoAcid)) +
    ggplot2::geom_col() +
    ggplot2::theme_bw() +
    ggplot2::theme(legend.position = "none")

  return(frequency_plot)
}
```

Group Discussion

1. Describe each function to each other in order - both what it does and which names you gave them and their variables.
2. Describe how you added the two packages as dependencies.
3. Discuss why it is a good idea to limit the number of dependencies your package has. When can't it always be avoided?

- Portability: Limiting dependencies can improve the portability of your package. If your package has fewer dependencies, it is more likely to work on different systems and environments without requiring additional installations or configuration, which make it easier for users to install and use.
- Efficiency: Having fewer dependencies can help your package run more efficiently and save time while installing and loading. This also makes it more likely to work with other packages.
- Reduced complexity: Every dependency adds complexity to your package. It's easier for developers to maintain the package.

4. Discuss the difference between adding an @importFrom package function tag to a function description compared to using package::function().

- It can be cumbersome to use package::function() repeatedly throughout your code, especially if you use many functions from the same package.