CentralDogmaFun

```r
library(CentralDogmaFun)
```

Group 10 GitHub repository

The following link is the GitHub repository of group 10:\ https://github.com/rforbiodatascience24/group10

Random_DNA_sequence

The function Random_DNA_sequence generates a random DNA sequence of a specified length. In the central dogma this can be viewed as the DNA replication process as it mimics the idea of DNA sequence formation. The Random_DNA_sequence function takes one argument called sequence_length. It samples from the four DNA nucleotides (A, T, G, C) with a sample size equal to sequence_length. The replace = TRUE allows each nucleotide to be sampled multiple times, creating a random sequence. This randomly generated nucleotide sequence (called DNA_sample) is taken and combined into a single string (a continuous DNA sequence) without any separators between characters.

If the input is set to 10 the function for instance returns a string of "AATGGATGAC". A seed is set, as the output is random, to make sure it is the same example shown in the code below.

```r
set.seed(101)
Random_DNA_sequence(10) # returns eg. "AATGGATGAC"
#> [1] "AATGGATGAC"
```

Transcription

The DNA_to_RNA function convert a DNA sequence into an RNA sequence by replacing thymine (T) with uracil (U) in the DNA input sequence. The process mimics the biological process of transcription, where DNA is transcribed to RNA, and thymine is substituted with uracil

In the DNA_to_RNA function a DNA sequence is taken as input, this is represented by a string of nucleotide letters (A, T, C and G). using the gsub function, all occurrences of "T" in the DNA sequence are replaced with "U" to reflect the substitution in nucleotides that happens during transcription. The resulting RNA sequence is then returned as output.

If "ATTGCG" is set as input, the function will return "AUUGCG". See the code for reference:

```r
DNA_to_RNA("ATTGCG") # returns "AUUCGC"
#> [1] "AUUGCG"
```

RNA_to_codon

The RNA_to_codon function converts a string of RNA seuqence into codons of 3 nucleotides. The procces mimics how the nucelotides of the mRNA sequence are being read into codons which then eventually is translated into amino acids in the portein seuqence.

The function takes as input a RNA sequence in a string plus the parameter start, which should be set to a default of = 1, unless another starting point than the beginning of the RNA sequence is wanted. The function then seperates the RNA string into a vector of strings containing the codon in each string.

If “UGGUCC” is set as input, e.i. RNA_to_codon(“UGGUCC”, start=1) the function will return c(“UGG”,“UCC”) as output where “UGG” and “UCC” is each a codon as the function begins at position 1.

```
RNA_to_codon("UGGUCC", start=1) # returns c("UGG","UCC")
#> [1] "UGG" "UCC"
```

codons_to_aa

The codons_to_aa function converts codons into amino acids. The process mimics the biological process of translation, where RNA is read in codons by the ribosome and the corrsponding amino acid is added to the growing peptide chain.

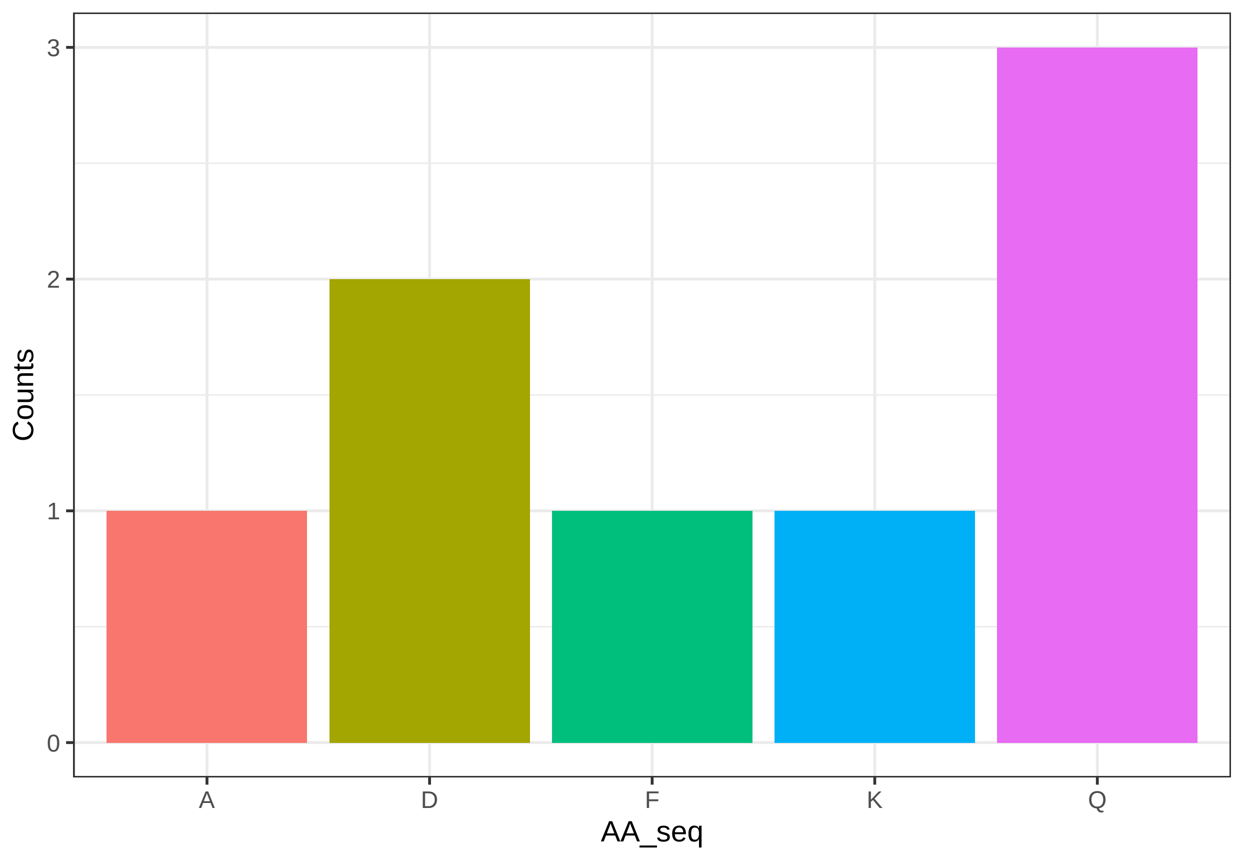
The function takes a vector of codons as strings as input. The strings are converted into the corresponding one-letter amino acid code using the codon table. The one-letter amino acid codes are combined into one string and returned as output. \ If c(“UUU”,“CAA”,“GAC”) is set as input, e.i. codons_to_aa(c(“UUU”,“CAA”,“GAC”)) the function will return “FQD” as output.

```
codons_to_aa(c("UUU","CAA","GAC")) # returns "FDQ"
#> [1] "FQD"
```

protein_plot

The protein_plot function takes a sequence of aminoacids (AA), counts the amount of each AA and plot them in a column-diagram/histogram. This is done in order to give an overview of the content of an AA-seq. The function works by first finding all unique AA in the sequence. Hereafter all the different proteins are counted in categories of the unique proteins. These counts are then used in a ggplot2::geom_col plot. Hereby the function is able to costumize the amount of columns (dependent on the unique AA’s). Hereby, this function is able to give a visualization of the contents in the end product of the central dogma.

```
protein_plot("FDQQDQKA") #plots a histogram of the AA contents. For this sequence there will be 5 colum
```

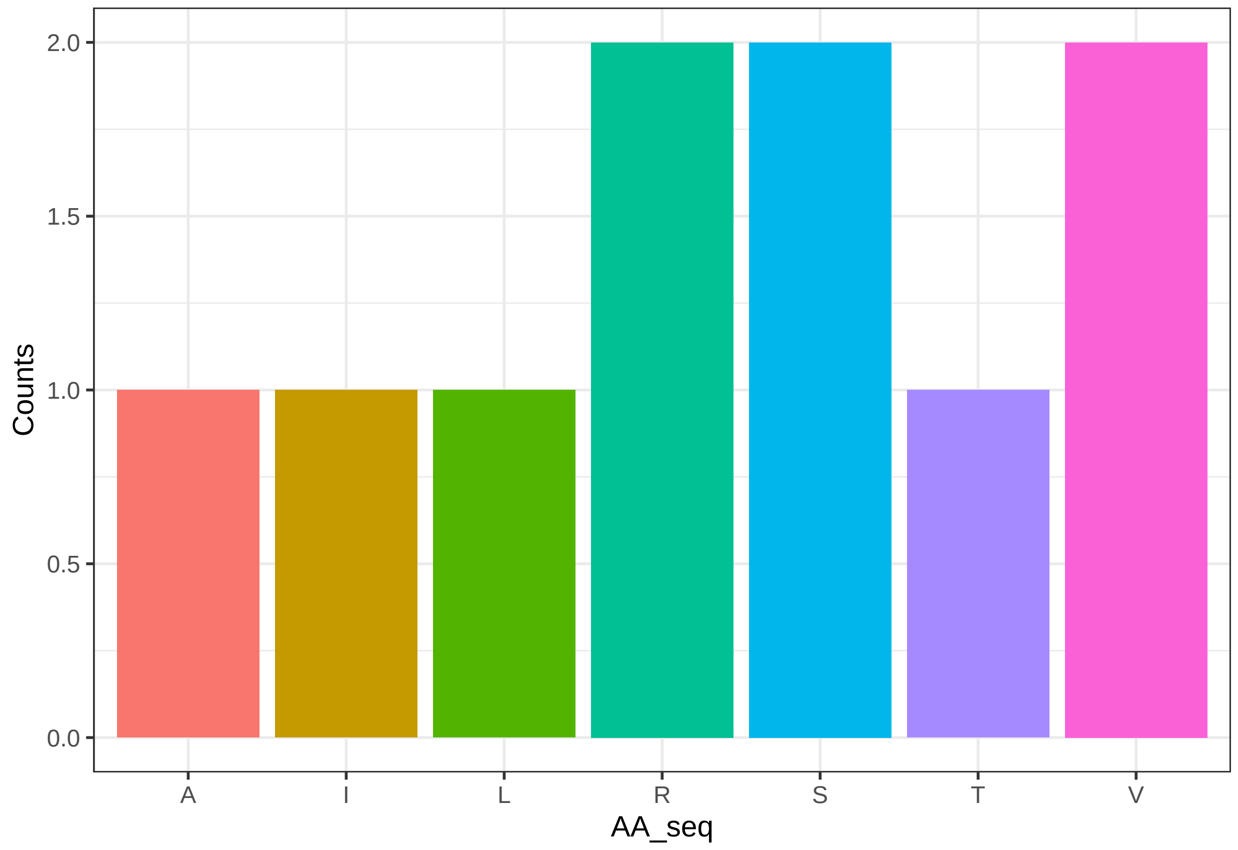


Use cases for the package

The package is intended to easily transcribe and translate a random or known DNA sequence, resembling the steps in the central dogma. The functions in the package are made so that they can be used sequentially in a pipeline and fully translate the DNA into protein and plot the count of the different amino acids.

```r
aa_count_plot <- Random_DNA_sequence(30) |>
  DNA_to_RNA() |>
  RNA_to_codon(start = 1) |>
  codons_to_aa() |>
  protein_plot()

aa_count_plot
```



Other functions to include

Here are some examples of how the package could be further extended with other types of functions relevant in the central dogma.

A replication fork function:

A function could be made that simulates the process of DNA unwinding and forming two complementary strands. It could take a DNA sequence and generate two new strands which mimicks the replication fork where the DNA splits into two complementary strands.

A DNA mutation function:\ Mutations can happen in the DNA replication, therefore a function that randomly introduces mutations into a DNA sequence could be relevant. This function could simulate different types of mutations such as: substitutions, insertions, or deletions.

A protein folding function:

The final step of the central dogma is the protein formation. Therefore, a relevant function could be one that takes an amino acid sequence and predicts a simple version of a protein structure - for instance based on types of foldings like alpha-helix and beta-sheet.

Discussion on limiting the number of dependencies

In order to make code more usable it is crucial to limit the number of dependencies.

The more dependencies a code is relying on, the more points of potential failure of the code arise. This is a consequence of code/packages being uploaded and then not updated. If a code is then built, and therefore dependent, on this package it might potentially stop working if said package is not updated along other packages needed in the code which might again then depend on each other.

Differences between loading packages

It is possible to call functions and packages in different manners. In general a package can be used by either loading or attaching. When loading the package you connect the specific function to the library you want to use it from. Where for using attaching you simply load the library and then call the function when needed in your code.

It might seem easier to use attach to call functions, however this also adds the risk of conflict if the name of a function is present in more than one of the libraries loaded.