HW Class 6 (R Functions)

Kathy Gu (A16362055)

Section 1: Improving analysis code by writing functions

A. Function to Normalize Columns in a DataFrame

To defining a function to normalize columns of a dataframe, I write the function below. This function 'normalize_column' is aim to normalize numeric columns in a dataframe. Normalization is a common preprocessing step in data analysis and machine learning, which scales numeric data to a [0, 1] range to ensure consistent measurements across features. It take the inputs of: 'df', a dataframe containing the data, and 'column_name', the name of the column to be normalized. The function returns a dataframe with the specified column normalized if it is numeric.

```
# Define a function to normalize a numeric column in a dataframe
normalize_column <- function(df, column_name) {
  # Ensure column is numeric
  if (is.numeric(df[[column_name]])) {
    # Calculate min and max, ignoring NA
    min_value <- min(df[[column_name]], na.rm = TRUE)
    max_value <- max(df[[column_name]], na.rm = TRUE)
    # Normalize if there's a range, otherwise set to 0
    if (max_value > min_value) {
      df[[column_name]] <- (df[[column_name]] - min_value) / (max_value - min_value)
    } else {
      df[[column_name]][!is.na(df[[column_name]])] <- 0
    }
  }
  # Return updated dataframe
  return(df)
}
```

```r
# Example usage:
# Create a sample dataframe
df <- data.frame(a=1:10, b=seq(200, 400, length=10), c=11:20, d=NA)
# Apply normalization to each column
df <- normalize_column(df, 'a')
df <- normalize_column(df, 'b')
df <- normalize_column(df, 'c')
df <- normalize_column(df, 'd')
# Display the modified dataframe
df
```

```
           a         b         c  d
1  0.0000000 0.0000000 0.0000000 NA
2  0.1111111 0.1111111 0.1111111 NA
3  0.2222222 0.2222222 0.2222222 NA
4  0.3333333 0.3333333 0.3333333 NA
5  0.4444444 0.4444444 0.4444444 NA
6  0.5555556 0.5555556 0.5555556 NA
7  0.6666667 0.6666667 0.6666667 NA
8  0.7777778 0.7777778 0.7777778 NA
9  0.8888889 0.8888889 0.8888889 NA
10 1.0000000 1.0000000 1.0000000 NA
```

> Q6. How would you generalize the original code above to work with any set of input protein structures?

To make the original code more versatile so it can handle any set of protein structures, I'd write a function that takes care of the repetitive steps like loading PDB files, processing them, and plotting the results. The function would accept a list of PDB identifiers and work through each one. This approach makes it easy to analyze different proteins just by supplying their PDB IDs.

B. Function to Analyze Protein Drug Interactions

To define a function to analyze protein drug interactions, I write the function below. This function, 'analyze_proteins', is aimed at analyzing and visualizing the B-factors of proteins from PDB files, which can indicate the flexibility of protein structures. B-factors are crucial for understanding protein dynamics, especially in drug interactions. It takes the input of 'pdb_ids', a vector of strings containing PDB IDs of the proteins to analyze. The function returns a list where each element contains the B-factors for a protein keyed by its PDB ID. This allows for easy access to data based on the protein identifier.

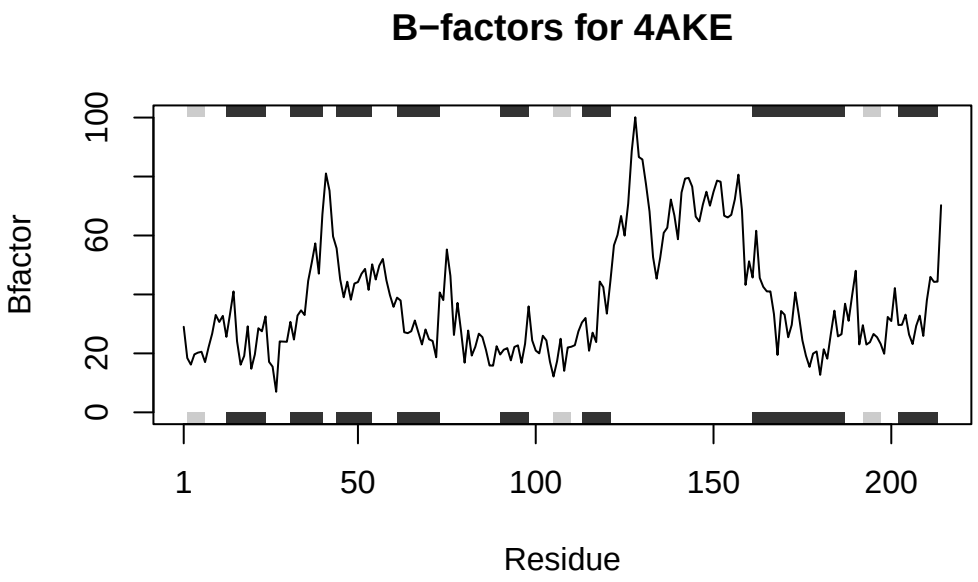
```
library(bio3d)
```

```
Warning: package 'bio3d' was built under R version 4.3.3
```

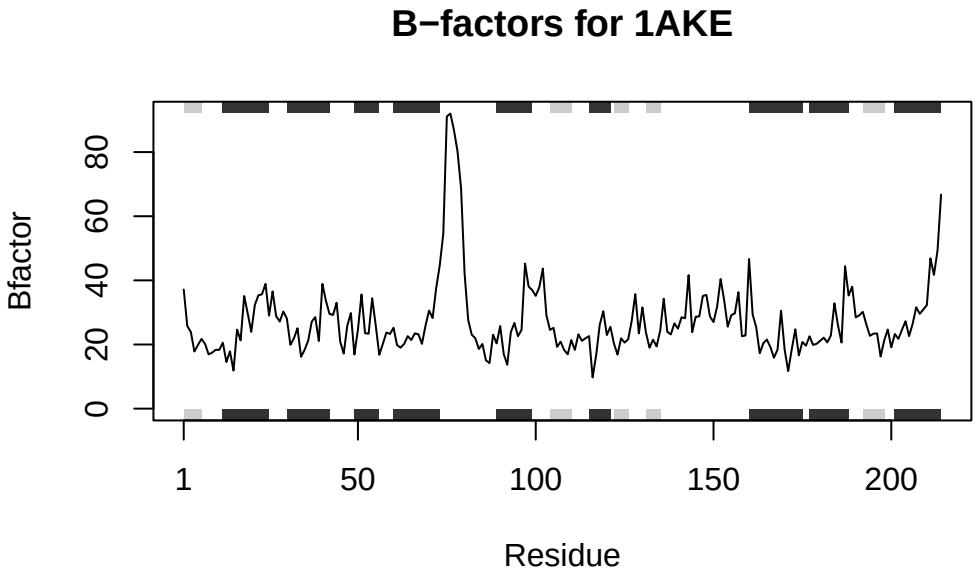
```
# Define function to process and plot B-factors from protein PDB files
analyze_proteins <- function(pdb_ids) {
  # Holds B-factor data
  b_factors_list <- list()
  # Process each protein by PDB ID
  for (pdb_id in pdb_ids) {
    protein <- read.pdb(pdb_id)
    protein_chain_A <- trim.pdb(protein, chain="A", elety="CA")
    # Extract B-factors
    b_factors <- protein_chain_A$atom$b
    # Store B-factors with PDB ID key
    b_factors_list[[pdb_id]] <- b_factors
    # Plot B-factors for visualization
    plotb3(b_factors, sse=protein_chain_A, typ="l", ylab="Bfactor", main=paste("B-factors
  }
  # Return list of B-factor data
  return(b_factors_list)
}

# Example of function usage
pdb_ids <- c("4AKE", "1AKE", "1E4Y")
b_factors_list <- analyze_proteins(pdb_ids)
```

```
 Note: Accessing on-line PDB file
```




```
Note: Accessing on-line PDB file
 PDB has ALT records, taking A only, rm.alt=TRUE
```



```
Note: Accessing on-line PDB file
```