Lab 6- Bio3d

Benjie Miao (A69026849)

Section 1: Improving analysis code by writing functions

A

A. Improve this regular R code by abstracting the main activities in your own new function. Note, we will go through this example together in the formal lecture. The main steps should entail running through the code to see if it works, simplifying to a core working code snippet, reducing any calculation duplication, and finally transferring your new streamlined code into a more useful function for you.

The original code:

```
# (A. Can you improve this analysis code?
df <- data.frame(a=1:10, b=seq(200,400,length=10),c=11:20,d=NA)
df$a <- (df$a - min(df$a)) / (max(df$a) - min(df$a))
df$b <- (df$b - min(df$b)) / (max(df$b) - min(df$b))
df$c <- (df$c - min(df$c)) / (max(df$c) - min(df$c))
df$d <- (df$d - min(df$d)) / (max(df$a) - min(df$d))
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
```

If we re-organize it with function, it could be more concise:

```
df <- data.frame(a=1:10, b=seq(200,400,length=10),c=11:20,d=NA)

normalize_column <- function(df, field) {
  df[,field] <<- (df[,field] - min(df[,field])) / (max(df[,field]) - min(df[,field])) # Ac
}

normalize_column(df, 'a')
normalize_column(df, 'b')
normalize_column(df, 'c')
normalize_column(df, 'd')
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
```

B. Bio3D

B. Next improve the below example code for the analysis of protein drug interactions by abstracting the main activities in your own new function. Then answer questions 1 to 6 below. It is recommended that you start a new Project in RStudio in a new directory and then install the bio3d package noted in the R code below (N.B. you can use the command install.packages("bio3d") or the RStudio interface to do this)

The original code:

```
# Can you improve this analysis code?
library(bio3d)
s1 <- read.pdb("4AKE") # kinase with drug
```

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

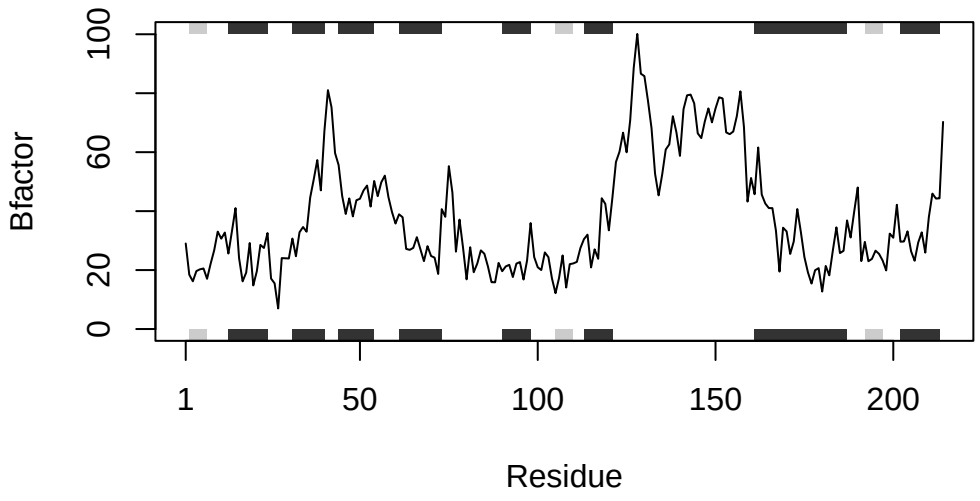
```
s2 <- read.pdb("1AKE") # kinase no drug
```

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

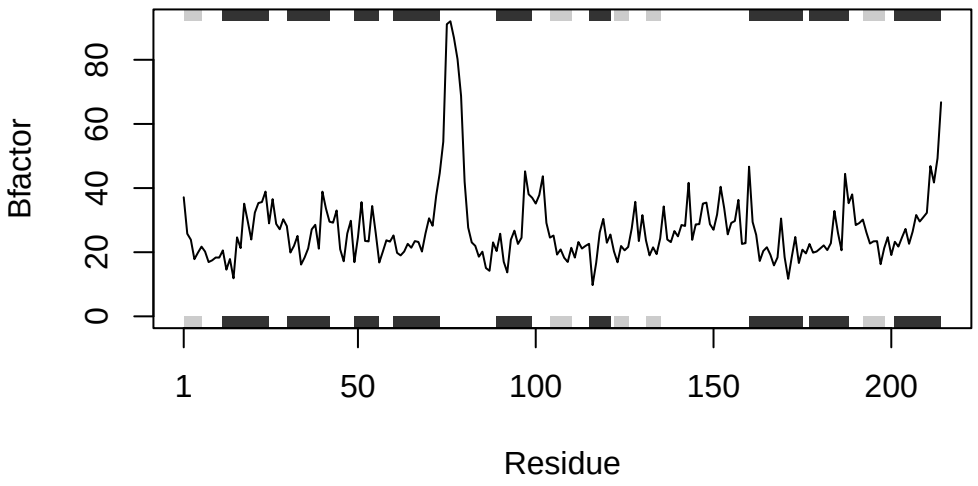
```
s3 <- read.pdb("1E4Y") # kinase with drug
```

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

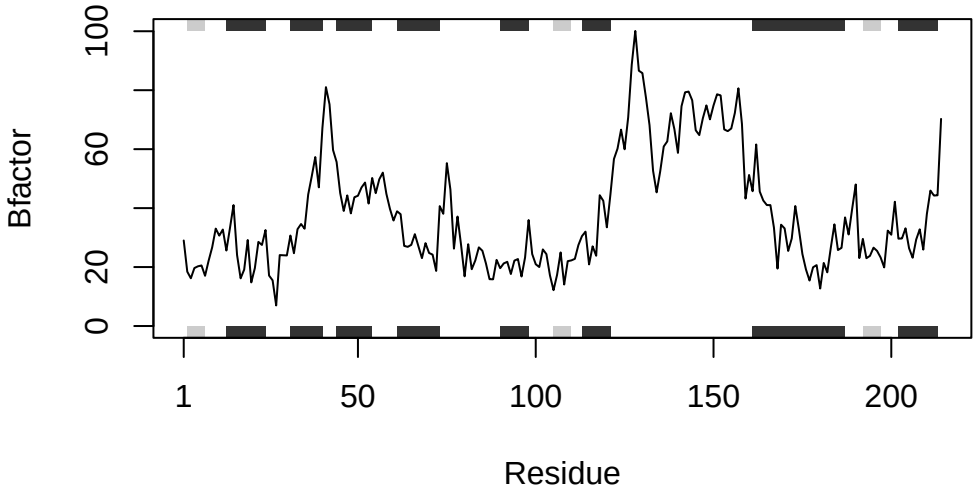
```
s1.chainA <- trim.pdb(s1, chain="A", elety="CA")
s2.chainA <- trim.pdb(s2, chain="A", elety="CA")
s3.chainA <- trim.pdb(s1, chain="A", elety="CA")
s1.b <- s1.chainA$atom$b
s2.b <- s2.chainA$atom$b
s3.b <- s3.chainA$atom$b
plotb3(s1.b, sse=s1.chainA, typ="l", ylab="Bfactor")
```




```
plotb3(s2.b, sse=s2.chainA, typ="l", ylab="Bfactor")
```




```
plotb3(s3.b, sse=s3.chainA, typ="l", ylab="Bfactor")
```



Again we can re-organize it into functions:

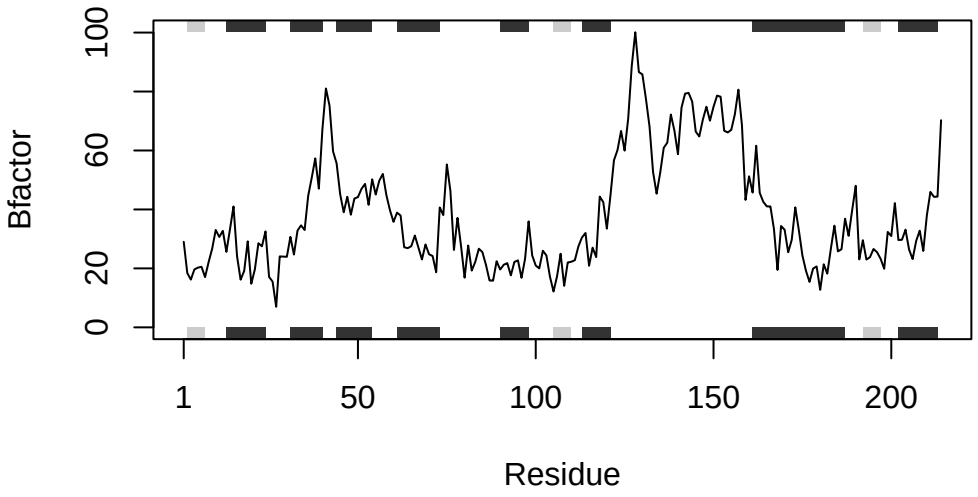
```
# Can you improve this analysis code?
library(bio3d)

#' Plot the pdb of a single protein
#'
#' @param The name of protein, as a string
#'
#' @return The plot object
#' @export
#'
#' @examples plot_pdb("4AKE")
plot_pdb <- function(name_of_protein) {
  s1 <- read.pdb(name_of_protein) # kinase with drug
  s1.chainA <- trim.pdb(s1, chain="A", elety="CA")
  s1.b <- s1.chainA$atom$b
  plotb3(s1.b, sse=s1.chainA, typ="l", ylab="Bfactor")
}

plot_pdb("4AKE")
```

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

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
C:\Users\bimia\AppData\Local\Temp\Rtmp2nbsEQ/4AKE.pdb exists. Skipping download
```

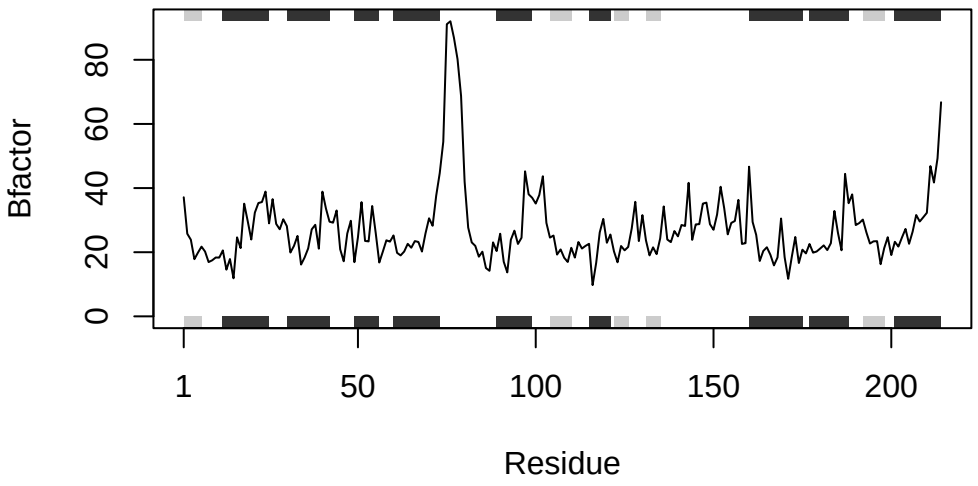



```
plot_pdb("1AKE")
```

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

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
C:\Users\bimia\AppData\Local\Temp\Rtmp2nbsEQ/1AKE.pdb exists. Skipping download
```

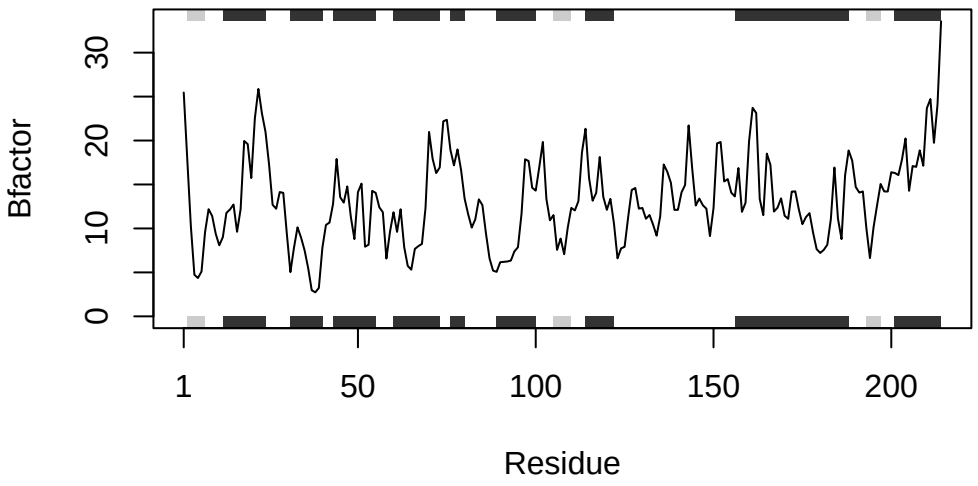
```
   PDB has ALT records, taking A only, rm.alt=TRUE
```




```
plot_pdb("1E4Y")
```

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

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
C:\Users\bimia\AppData\Local\Temp\Rtmp2nbsEQ/1E4Y.pdb exists. Skipping download
```



Questions:

Q1: What type of object is returned from the read.pdb() function?

```
s1 <- read.pdb("4AKE")
```

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

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
C:\Users\bimia\AppData\Local\Temp\Rtmp2nbsEQ/4AKE.pdb exists. Skipping download
```

```
class(s1)
```

```
[1] "pdb" "sse"
```

It is a 'pdb sse' object.

Q2. What does the trim.pdb() function do?

```
help(trim.pdb)
```

It produces a new smaller PDB object, containing a subset of atoms, from a given larger PDB object.

** Q3. What input parameter would turn off the marginal black and grey rectangles in the plots and what do they represent in this case? **

```
help("plotb3")
```

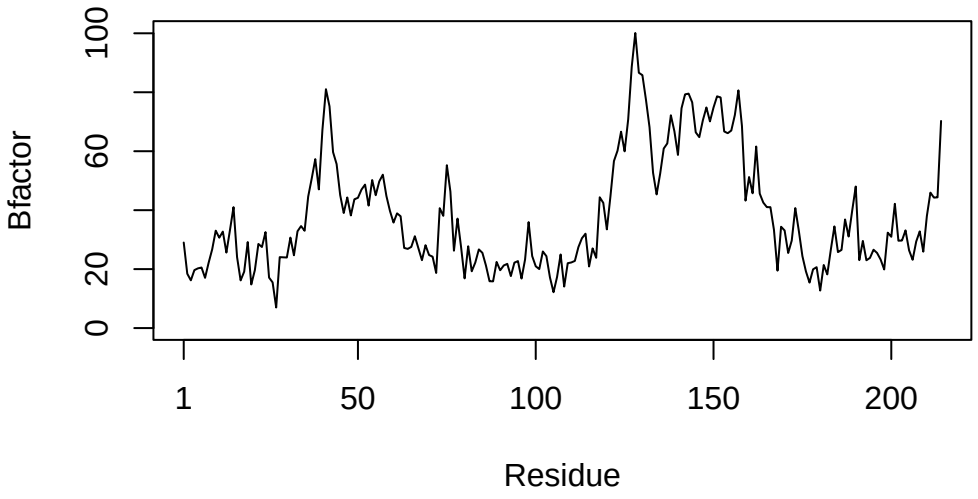
From the document, the `top` and `bot` parameters will control it.

```
s1 <- read.pdb("4AKE")
```

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

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
C:\Users\bimia\AppData\Local\Temp\Rtmp2nbsEQ/4AKE.pdb exists. Skipping download
```

```
s1.chainA <- trim.pdb(s1, chain="A", elety="CA")
s1.b <- s1.chainA$atom$b
plotb3(s1.b, sse=s1.chainA, typ="l", ylab="Bfactor", top=FALSE, bot=FALSE)
```



** Q4. What would be a better plot to compare across the different proteins? **

We can plot different proteins in one coordinate:

```
#' Get the B factor of a single protein
#'
#' @param The name of protein, as a string
#'
#' @return The factor, as an array
#' @export
#'
#' @examples get  _pdb("4AKE")
get_pdb <- function(name_of_protein) {
  s1 <- read.pdb(name_of_protein)
  s1.chainA <- trim.pdb(s1, chain="A", elety="CA")
  s1.b <- s1.chainA$atom$b
  return(s1.b)
}

a <- get_pdb("4AKE")
```

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

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
C:\Users\bimia\AppData\Local\Temp\Rtmp2nbsEQ/4AKE.pdb exists. Skipping download
```

```
b <- get_pdb("1AKE")
```

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

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
C:\Users\bimia\AppData\Local\Temp\Rtmp2nbsEQ/1AKE.pdb exists. Skipping download
```

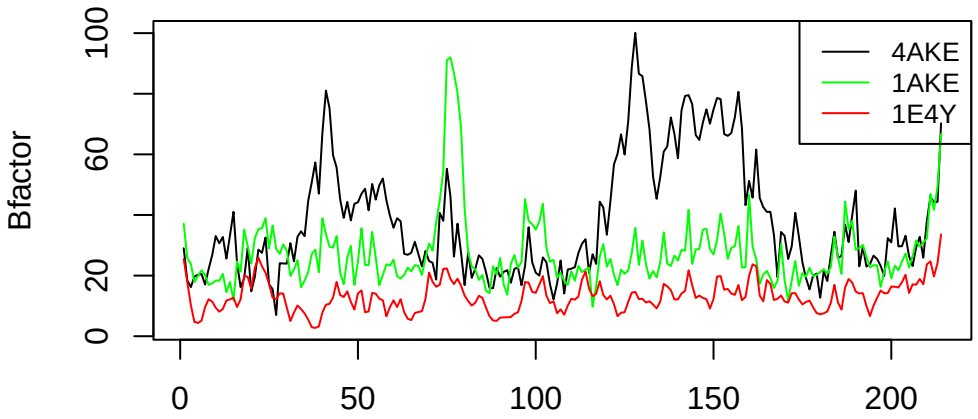
```
   PDB has ALT records, taking A only, rm.alt=TRUE
```

```
c <- get_pdb("1E4Y")
```

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

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
C:\Users\bimia\AppData\Local\Temp\Rtmp2nbsEQ/1E4Y.pdb exists. Skipping download
```

```
legends <- c("4AKE", "1AKE", "1E4Y")
df <- data.frame(a, b, c)
colnames(df) <- legends
colors <- c('black','green','red')
# Since we have this df, we can plot then in one axis
matplot(df, type = "l", lty = "solid",
          pch=NA_integer_, col = colors, ylab='Bfactor')
legend("topright", legend = legends, col=colors,lty = "solid", cex=0.8)
```



** Q5. Which proteins are more similar to each other in their B-factor trends. How could you quantify this? HINT: try the rbind(), dist() and hclust() functions together with a resulting dendrogram plot. Look up the documentation to see what each of these functions does. **

```
a <- get_pdb("4AKE")
```

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

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
```

```
C:\Users\bimia\AppData\Local\Temp\Rtmp2nbsEQ/4AKE.pdb exists. Skipping download
```

```
b <- get_pdb("1AKE")
```

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

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
C:\Users\bimia\AppData\Local\Temp\Rtmp2nbsEQ/1AKE.pdb exists. Skipping download
```

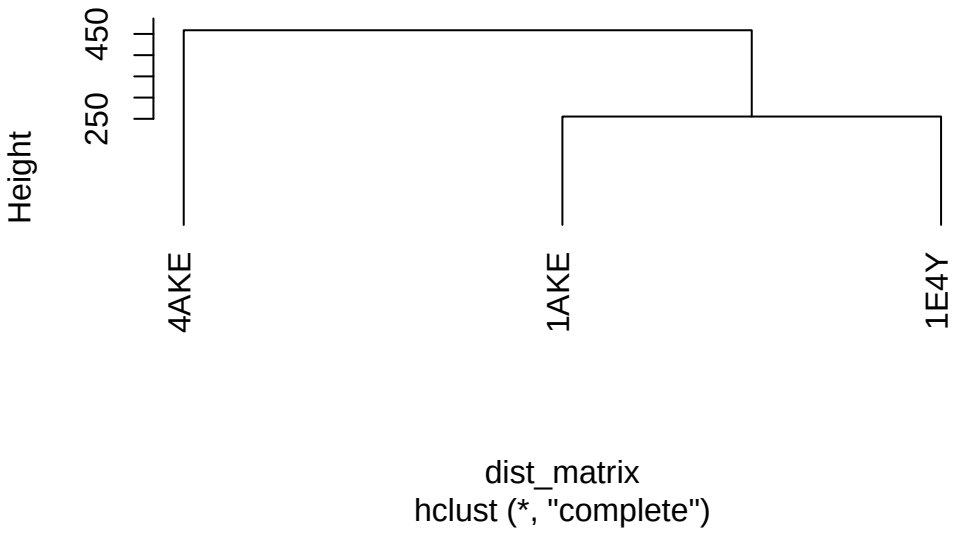
```
   PDB has ALT records, taking A only, rm.alt=TRUE
```

```
c <- get_pdb("1E4Y")
```

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

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
C:\Users\bimia\AppData\Local\Temp\Rtmp2nbsEQ/1E4Y.pdb exists. Skipping download
```

```
df <- data.frame(a, b, c)
legends <- c("4AKE", "1AKE", "1E4Y")
colnames(df) <- legends
dist_matrix <- dist(rbind(a, b, c), diag=TRUE, upper=TRUE)
hc1 <- hclust(dist_matrix)
plot(hc1, labels = legends, hang = -1, main = "")
```



1AKE and 1E4Y are more similar.

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

We can construct two levels of function to make it automatic.

```
get_pdb <- function(name_of_protein) {
  s1 <- read.pdb(name_of_protein) # kinase with drug
  s1.chainA <- trim.pdb(s1, chain="A", elety="CA")
  s1.b <- s1.chainA$atom$b
  return(s1.b)
}

hclust_bio <- function(input_protein) {
  # calculate B factor for each protein

  f <- lapply(input_protein, FUN=get_pdb)
  # calculate the distance matrix, and we need to use the transpose one
  all_protein_feature <- do.call("rbind",f)
  # get thr dist matrix
  dist_matrix <- dist(all_protein_feature)
  # hierarchical clustering
```

```
  hc1 <- hclust(dist_matrix)
  plot(hc1, labels = input_protein, hang = -1)
}

input_protein <- c("4AKE", "1AKE", "1E4Y")
hclust_bio(input_protein)
```

```
  Note: Accessing on-line PDB file

Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
C:\Users\bimia\AppData\Local\Temp\Rtmp2nbsEQ/4AKE.pdb exists. Skipping download

  Note: Accessing on-line PDB file

Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
C:\Users\bimia\AppData\Local\Temp\Rtmp2nbsEQ/1AKE.pdb exists. Skipping download

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

Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
C:\Users\bimia\AppData\Local\Temp\Rtmp2nbsEQ/1E4Y.pdb exists. Skipping download
```