Homework 6

Job Rocha PID A59023124

Section 1: Improving analysis code by writing functions 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.

```
# (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$b - min(df$a)) / (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))
```

Improved code

```
calculate.numerator <- function(x, y){ return (x - min(y)) }
calculate.denominator <- function(x, y){ return (max(x) - min(y)) }

analysis.function <- function(DF, cols){
  # Takes a dataframe (DF) and a vector of 4 columns (cols)
  # Returns a vector of doubles
  return (calculate.numerator(DF[,cols[1]], DF[, cols[2]]) / calculate.denominator(DF[, co
}

improved.function <- function(DF, columns){
  # Takes a dataframe DF and a list (columns) of vectors where each vector has 4 column na
  # Returns a datafrmae with columns's keys as columns.
  return (as.data.frame(lapply(columns,FUN=function(cols){analysis.function(DF, cols)})))
}
```

```
# Create input dataframe
df <- data.frame(a=1:10, b=seq(200,400,length=10),c=11:20,d=NA)
# Select what columns to use for each analysis
analysis <- list(
                  "a"=c("a", "a", "a", "a"),
                  "b"=c("b", "a", "b", "b"),
                  "c"=c("c", "c", "c", "c"),
                  "d"=c("d", "d", "a", "d")
                  )

#Run analysis
improved.function(df, analysis)
```

```
          a        b         c  d
1  0.0000000 0.995000 0.0000000 NA
2  0.1111111 1.106111 0.1111111 NA
3  0.2222222 1.217222 0.2222222 NA
4  0.3333333 1.328333 0.3333333 NA
5  0.4444444 1.439444 0.4444444 NA
6  0.5555556 1.550556 0.5555556 NA
7  0.6666667 1.661667 0.6666667 NA
8  0.7777778 1.772778 0.7777778 NA
9  0.8888889 1.883889 0.8888889 NA
10 1.0000000 1.995000 1.0000000 NA
```

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).

```
#install.packages("bio3d")
```

Then run through the code to see if it works, fix any copy/paste errors before simplifying to a core working code snippet, reducing any calculation duplication, and finally transferring it into a more useful function for you.

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

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

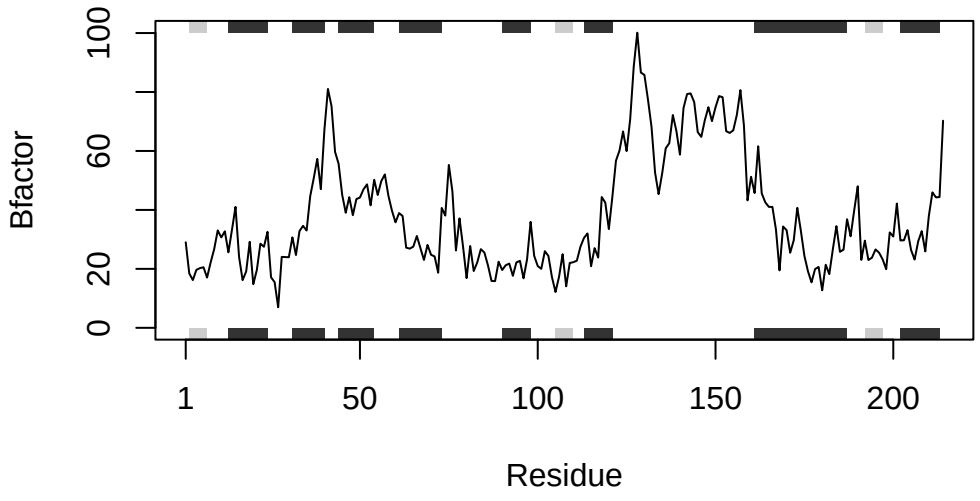
```
s2 <- read.pdb("1AKE")  # kinase no drug
```

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

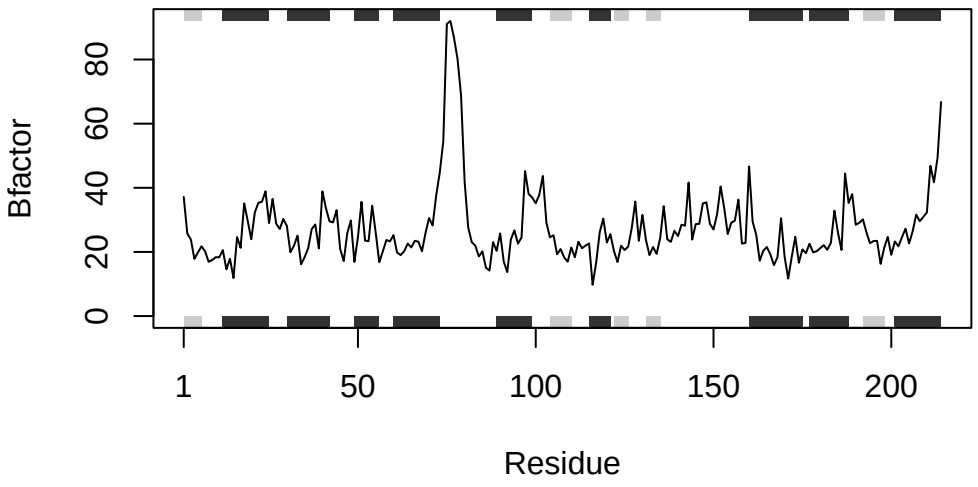
```
s3 <- read.pdb("1E4Y")  # kinase with drug
```

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

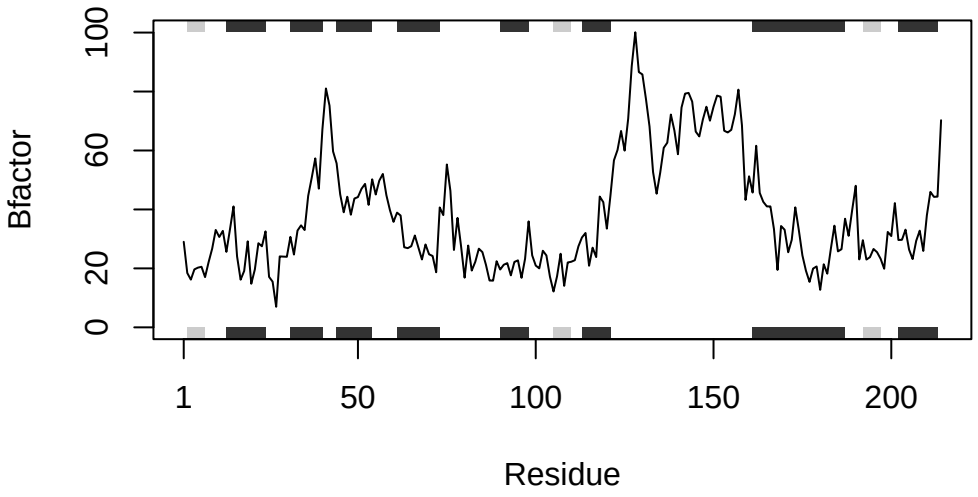
```
s1.chainA <- trim.pdb(s1, chain="A", elety="CA")
s2.chainA <- trim.pdb(s2, chain="A", elety="CA")
s3.chainA <- trim.pdb(s1, chain="A", elety="CA")
s1.b <- s1.chainA$atom$b
s2.b <- s2.chainA$atom$b
s3.b <- s3.chainA$atom$b
plotb3(s1.b, sse=s1.chainA, typ="l", ylab="Bfactor")
```



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




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



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

```
str(s1)
```

```
List of 8
 $ atom  :'data.frame': 3459 obs. of  16 variables:
  ..$ type  : chr [1:3459] "ATOM" "ATOM" "ATOM" "ATOM" ...
  ..$ eleno : int [1:3459] 1 2 3 4 5 6 7 8 9 10 ...
  ..$ elety : chr [1:3459] "N" "CA" "C" "O" ...
  ..$ alt   : chr [1:3459] NA NA NA NA ...
  ..$ resid : chr [1:3459] "MET" "MET" "MET" "MET" ...
  ..$ chain : chr [1:3459] "A" "A" "A" "A" ...
  ..$ resno : int [1:3459] 1 1 1 1 1 1 1 2 2 ...
  ..$ insert: chr [1:3459] NA NA NA NA ...
  ..$ x     : num [1:3459] -10.93 -9.9 -9.17 -9.8 -10.59 ...
  ..$ y     : num [1:3459] -24.9 -24.4 -23.3 -22.3 -24 ...
  ..$ z     : num [1:3459] -9.52 -10.48 -9.81 -9.35 -11.77 ...
  ..$ o     : num [1:3459] 1 1 1 1 1 1 1 1 1 1 ...
  ..$ b     : num [1:3459] 41.5 29 27.9 26.4 34.2 ...
  ..$ segid : chr [1:3459] NA NA NA NA ...
  ..$ elesy : chr [1:3459] "N" "C" "C" "O" ...
  ..$ charge: chr [1:3459] NA NA NA NA ...
```

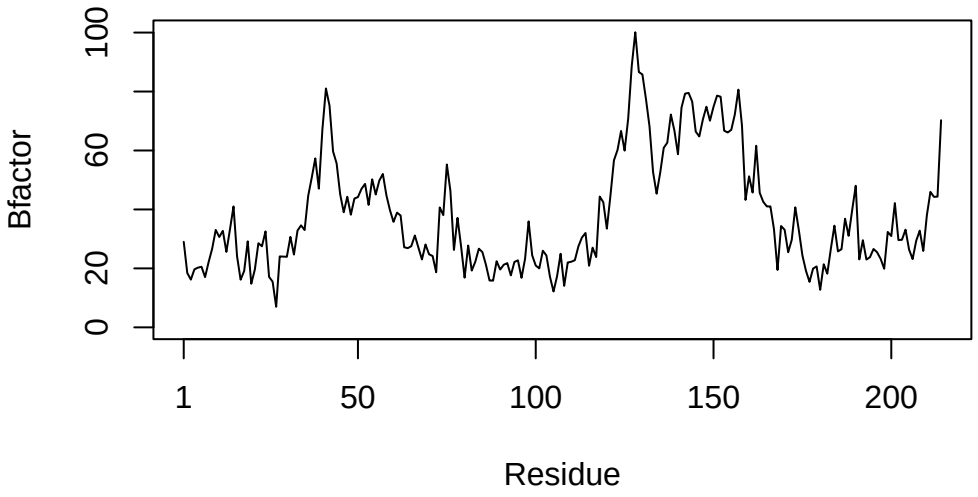
```
$ xyz   : 'xyz' num [1, 1:10377] -10.93 -24.89 -9.52 -9.9 -24.42 ...
$ seqres: Named chr [1:428] "MET" "ARG" "ILE" "ILE" ...
 ..- attr(*, "names")= chr [1:428] "A" "A" "A" "A" ...
$ helix :List of 4
 ..$ start: Named num [1:19] 13 31 44 61 75 90 113 161 202 13 ...
 .. ..- attr(*, "names")= chr [1:19] "" "" "" "" ...
 ..$ end  : Named num [1:19] 24 40 54 73 77 98 121 187 213 24 ...
 .. ..- attr(*, "names")= chr [1:19] "" "" "" "" ...
 ..$ chain: chr [1:19] "A" "A" "A" "A" ...
 ..$ type : chr [1:19] "5" "1" "1" "1" ...
$ sheet :List of 4
 ..$ start: Named num [1:14] 192 105 2 81 27 123 131 192 105 2 ...
 .. ..- attr(*, "names")= chr [1:14] "" "" "" "" ...
 ..$ end  : Named num [1:14] 197 110 7 84 29 126 134 197 110 7 ...
 .. ..- attr(*, "names")= chr [1:14] "" "" "" "" ...
 ..$ chain: chr [1:14] "A" "A" "A" "A" ...
 ..$ sense: chr [1:14] "0" "1" "1" "1" ...
$ calpha: logi [1:3459] FALSE TRUE FALSE FALSE FALSE FALSE ...
$ remark:List of 1
 ..$ biomat:List of 4
 .. ..$ num   : int 1
 .. ..$ chain :List of 1
 .. .. ..$ : chr [1:2] "A" "B"
 .. ..$ mat   :List of 1
 .. .. ..$ :List of 1
 .. .. .. ..$ A B: num [1:3, 1:4] 1 0 0 0 1 0 0 0 1 0 ...
 .. ..$ method: chr "AUTHOR"
$ call  : language read.pdb(file = "4AKE")
- attr(*, "class")= chr [1:2] "pdb" "sse"
```

Q2. What does the trim.pdb() function do? **Produce a new smaller PDB object, containing a subset of atoms, from a given larger PDB object.**

```
?trim.pdb
```

Q3. What input parameter would turn off the marginal black and grey rectangles in the plots and what do they represent in this case? **Parameter 'sse'. The black and grey rectangles represent secondary structures, alpha helix and b-sheet.**

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



Q4. What would be a better plot to compare across the different proteins? **Probably some kind of tree/dendogram where we can see how similar are the proteins to each other based on the hierarchy of the branches**

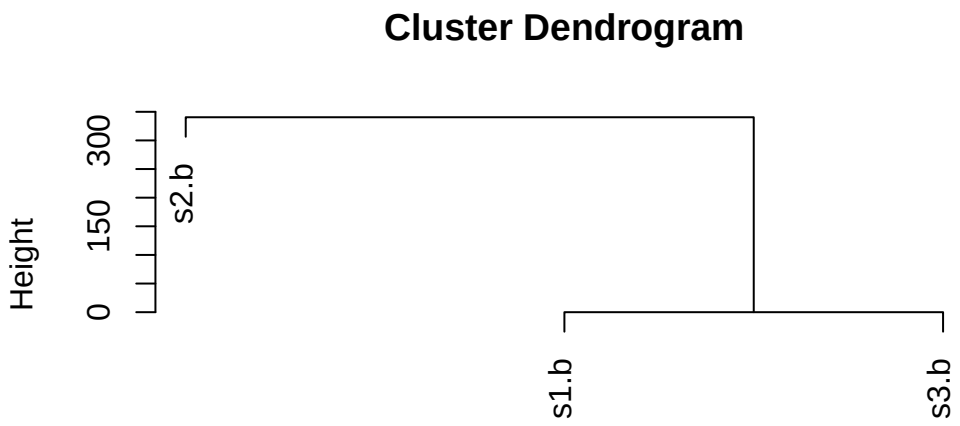
Q5. Which proteins are more similar to each other in their B-factor trends. How could you quantify this? HINT: try the rbind(), dist() and hclust() functions together with a resulting dendrogram plot. Look up the documentation to see what each of these functions does.

Answer Which proteins are more similar to each other in their B-factor trends?. *Proteins s1.b and s3.b are more similar to each other, as they have a low dissimilarity value as show by the dendogram below.*

How could you quantify this? *Using a dissimilarity/distance metric using the dist() function.*

Look up the documentation to see what each of these functions does:
- rbind: Combines multiple vectors by rows. Make each vector a row in a 2D-shaped array (matrix, data.frame, etc).
- dist: Computes a distance metric between each row of a dataframe.
- hclust: Computes a hierarchical cluster analysis on a set of dissimilarities.

```
hc <- hclust( dist( rbind(s1.b, s2.b, s3.b) ) )
plot(hc)
```



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

```
library(bio3d)

plot.3d.chain.from.id <- function(id, chain = "A", elety = "CA", type = "l", ylab = "Bfact
  ################## Documentation ######################
  # This function takes an pdb ID, selects specified protein chain, extracts chain's atoms

  ### Usage ###
  # plot.3d.chain.from.id("4AKE", chain="A", elety="CA", type="l", ylab="Bfactor")

  ### Inputs ###
  # id: String. Indicates the id of the protein
  # chain: Single character. Selects protein chain to use. Default: 'A'
  # elety: A character vector of atom names. Default: 'CA'
  # type: One-character string giving the type of plot desired. Default: 'l' (lines)
  # ylab: A label for the y axis. Default: 'Bfactor'

  ### Output ###
  #A standard scatter plot with secondary structure in the marginal regions.
```

```
  #Read pdb file based on input id
  S <- read.pdb(id)

  #Subsets pdb file to include only specified chain.
  S.chain <- trim.pdb(S, chain = chain, elety = elety)

  #Extract atoms from chain
  S.atoms <- S.chain$atom$b

  #Plot
  return( plotb3(S.atoms, sse = S.chain, typ = type, ylab = ylab) )
}

#TEST
ID <- "4AKE"
plot.3d.chain.from.id(ID)
```

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

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
/var/folders/7s/_b0b7yy112b4s9lgyy25dt48z59c1c/T//RtmpQ4PL5a/4AKE.pdb exists.
Skipping download
```

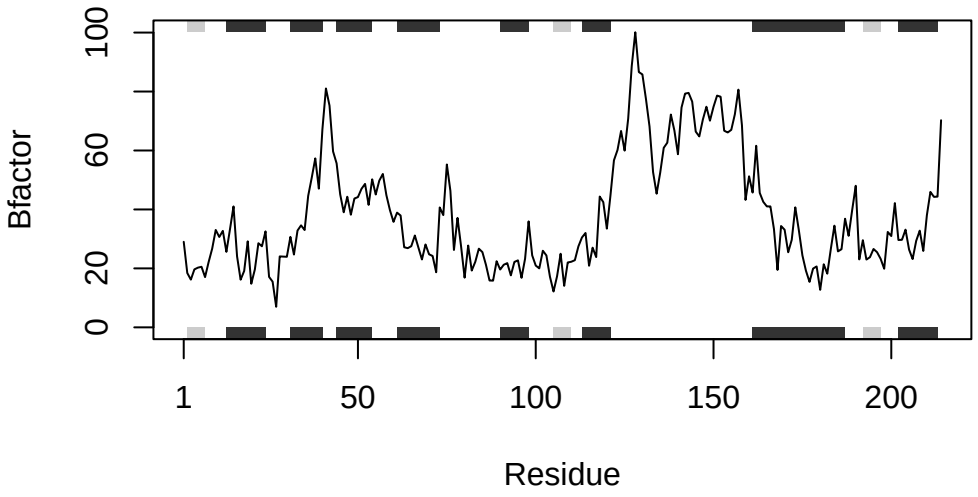
```
#MULTI TEST
IDs <- c("4AKE", "1AKE", "1E4Y")
sapply(IDs, plot.3d.chain.from.id)
```

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

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
/var/folders/7s/_b0b7yy112b4s9lgyy25dt48z59c1c/T//RtmpQ4PL5a/4AKE.pdb exists.
Skipping download
```

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

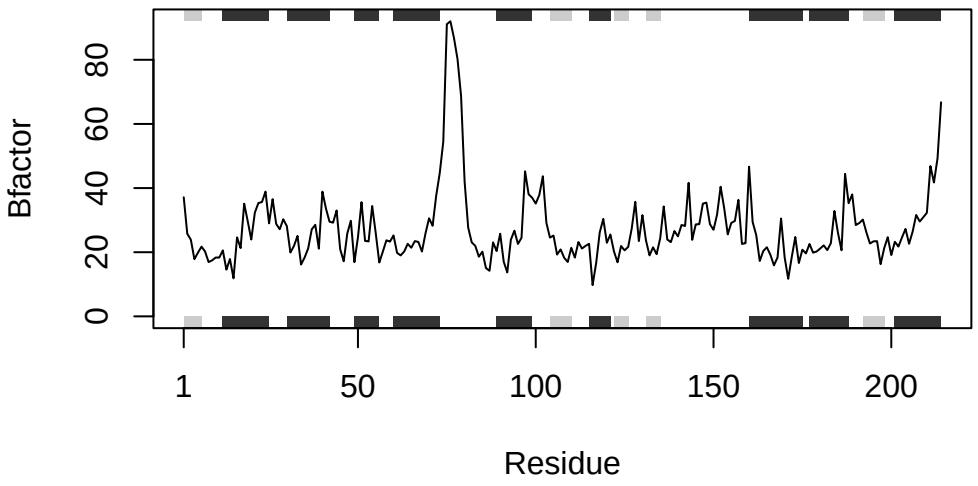
```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
/var/folders/7s/_b0b7yy112b4s9lgyy25dt48z59c1c/T//RtmpQ4PL5a/1AKE.pdb exists.
Skipping download
```

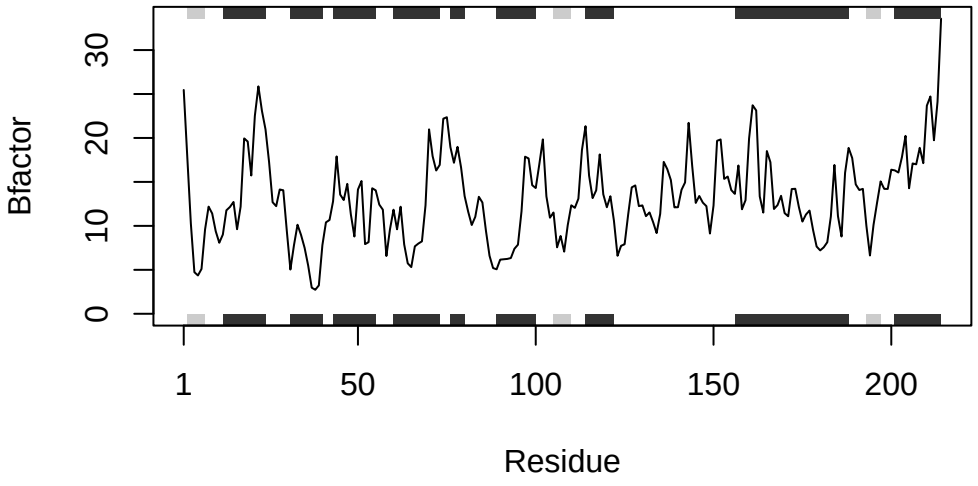



```
   PDB has ALT records, taking A only, rm.alt=TRUE

  Note: Accessing on-line PDB file

Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
/var/folders/7s/_b0b7yy112b4s9lgyy25dt48z59c1c/T//RtmpQ4PL5a/1E4Y.pdb exists.
Skipping download
```


Bfactor
0
20
40
60
80
1
50
100
150
200
Residue


Bfactor
0
10
20
30
1
50
100
150
200
Residue

```
$`4AKE`
NULL
```

```
$`1AKE`
NULL

$`1E4Y`
NULL
```