HW 6

Kristiana Wong A16281367

##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.

Lets take a look at our data frame before we dive into the code.

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

```
   a        b  c  d
1  1 200.0000 11 NA
2  2 222.2222 12 NA
3  3 244.4444 13 NA
4  4 266.6667 14 NA
5  5 288.8889 15 NA
6  6 311.1111 16 NA
7  7 333.3333 17 NA
8  8 355.5556 18 NA
9  9 377.7778 19 NA
10 10 400.0000 20 NA
```

First, we need to check if the code is working properly:

```
#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$b) - min(df$d))
```

This function essentially normalizes all the datasets to its respective column. We know this is working since the first value should always be 0 (base value) and the final value should be 1 (the maximum value).

Next, lets improve the code by condensing it into a singular line of code. I will be calling this function improve, and it is essentially a normalization function for each column. The input, here called column, will be each of the columns in the df data frame. The function will be applied to each column. In order to use this, you can do the following:

```
#A. Can you improve this analysis code?
improve <- function(column){
  column <- (column - min(column)) / (max(column) - min(column))
}


columns <- list(df$a, df$b, df$c, df$d)
for (col in columns){
  print(improve(col))
}
```

```
[1] 0.0000000 0.1111111 0.2222222 0.3333333 0.4444444 0.5555556 0.6666667
[8] 0.7777778 0.8888889 1.0000000
[1] 0.0000000 0.1111111 0.2222222 0.3333333 0.4444444 0.5555556 0.6666667
[8] 0.7777778 0.8888889 1.0000000
[1] 0.0000000 0.1111111 0.2222222 0.3333333 0.4444444 0.5555556 0.6666667
[8] 0.7777778 0.8888889 1.0000000
[1] NA NA NA NA NA NA NA NA NA NA
```

This will generate the normalization similar to the original code. I've created a loop function to make the code more simplistic.

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).

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

Check if code works:

```
library(bio3d)
s1 <- read.pdb("4AKE") # kinase with drug
```

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

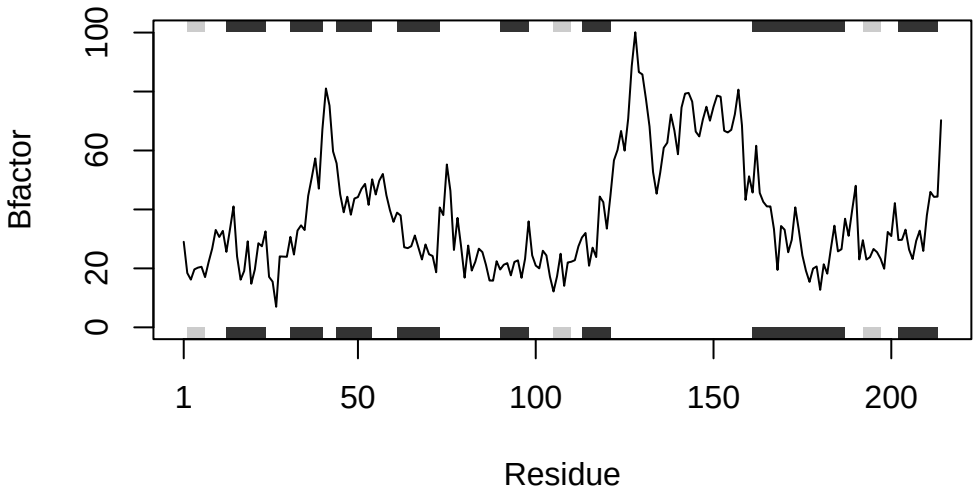
```
s2 <- read.pdb("1AKE") # kinase no drug
```

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

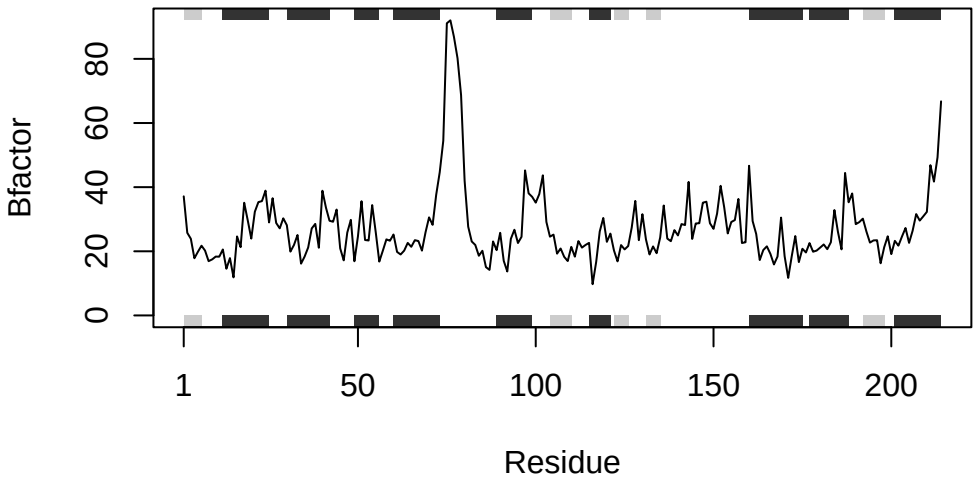
```
s3 <- read.pdb("1E4Y") # kinase with drug
```

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

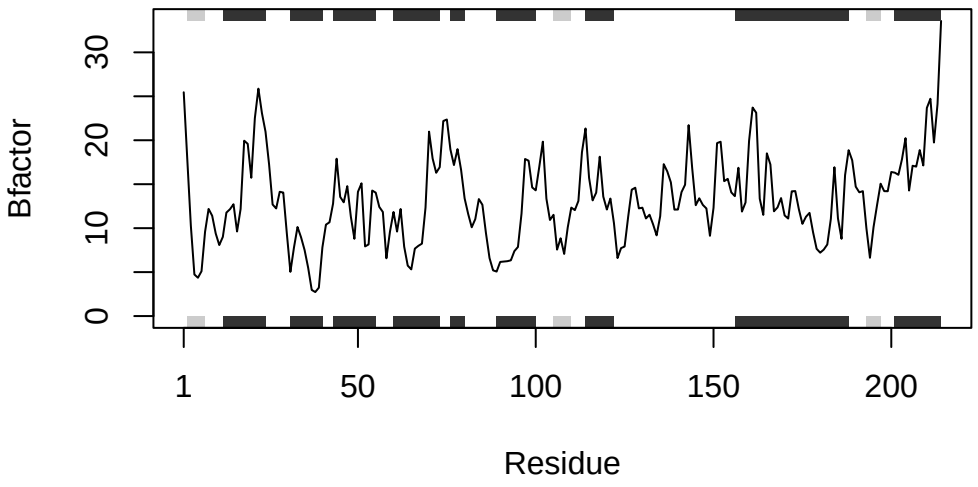
```
s1.chainA <- trim.pdb(s1, chain="A", elety="CA")
s2.chainA <- trim.pdb(s2, chain="A", elety="CA")
s3.chainA <- trim.pdb(s3, chain="A", elety="CA")
s1.b <- s1.chainA$atom$b
s2.b <- s2.chainA$atom$b
s3.b <- s3.chainA$atom$b
plotb3(s1.b, sse=s1.chainA, typ="l", ylab="Bfactor")
```




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



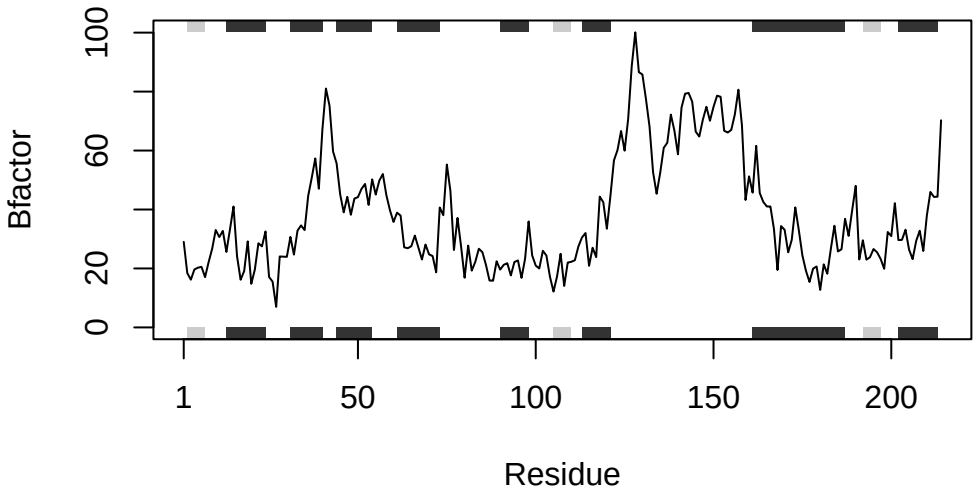
```
plotb3(s3.b, sse=s3.chainA, typ="l", ylab="Bfactor")
```



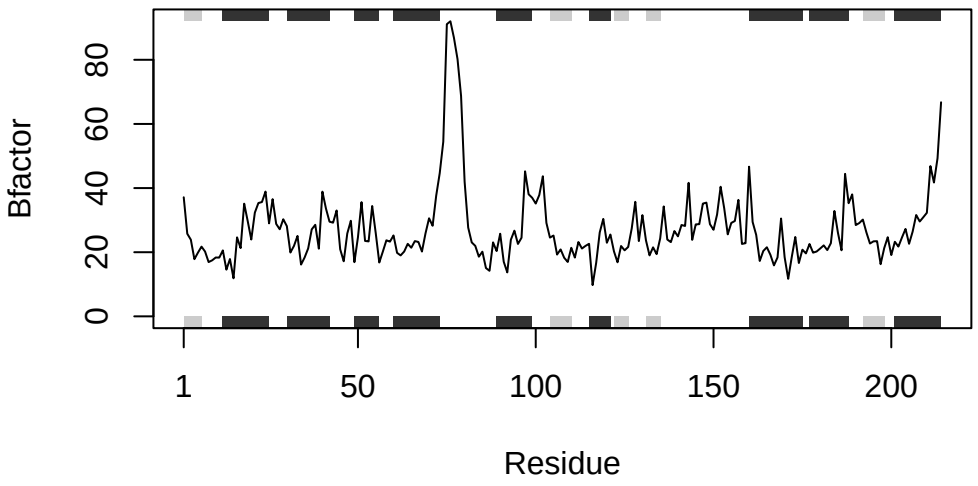
Next, improve the code. Here, the inputs will be the 3 different kinases: s1, s2, and s3. I will be creating 4 functions: 1. To read the kinase from the PDB file 2. To trim the file, essentially selecting a subset of the data available from the kinases 3. Creating a function to group the previous into the selected residue 4. Creating a plot of the residue vs the Bfactor

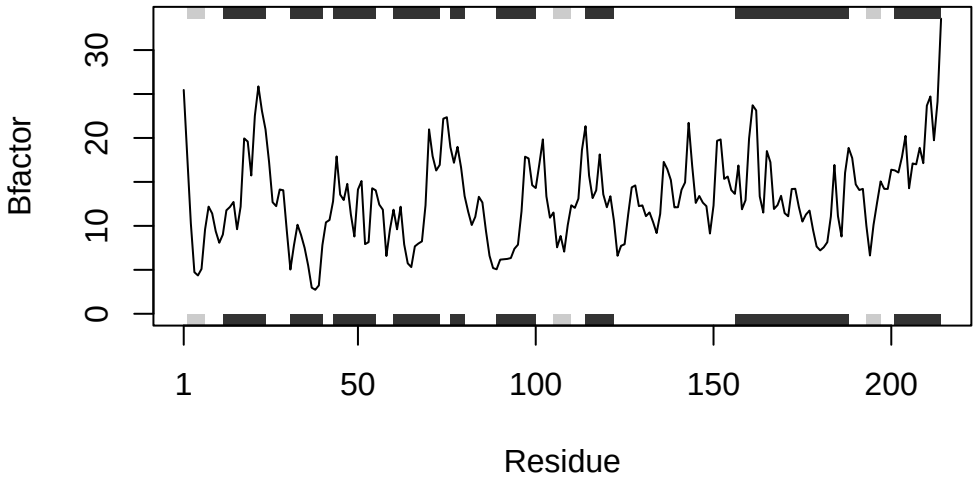
```
# Can you improve this analysis code?
library(bio3d)

analyze_kinase <- function(kinase){
  s <- read.pdb(kinase)
  s.chainA <- trim.pdb(s, chain="A", elety="CA")
  s.b <- s.chainA$atom$b
  plotb3(s.b, sse=s.chainA, typ="l", ylab="Bfactor")
}

kinases <- list("4AKE","1AKE","1E4Y")
for (residue in kinases){
  analyze_kinase(residue)
}
```

```
  Note: Accessing on-line PDB file

Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
/var/folders/wp/np_tf1817m753wgp9s4656400000gn/T//Rtmpyg6Znw/4AKE.pdb exists.
Skipping download

  Note: Accessing on-line PDB file

Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
/var/folders/wp/np_tf1817m753wgp9s4656400000gn/T//Rtmpyg6Znw/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/wp/np_tf1817m753wgp9s4656400000gn/T//Rtmpyg6Znw/1E4Y.pdb exists.
Skipping download
```





Q1. What type of object is returned from the read.pdb() function? It returns a note that it is reading from an online Protein Data Bank file.

Q2. What does the trim.pdb() function do? trim.pdb returns a subset of a given larger PDB object, essentially narrowing down the file to a certain subset of the data.

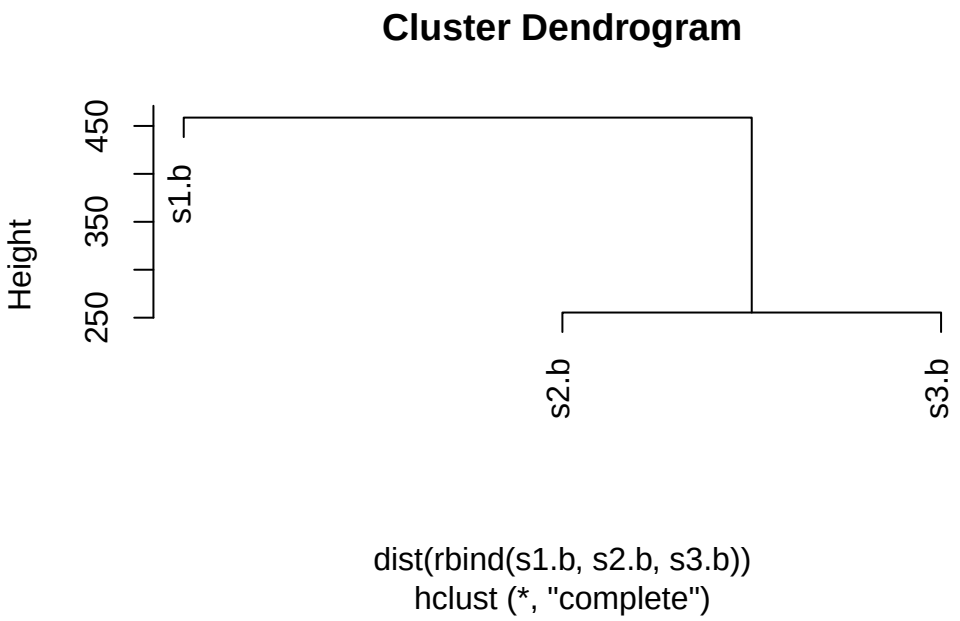
Q3. What input parameter would turn off the marginal black and grey rectangles in the plots and what do they represent in this case? Removing the 'sse=s.chainA' input removes the black and grey rectangles. This represents the secondary structure object.

Q4. What would be a better plot to compare across the different proteins? Likely a panel of plots to compare across the proteins, such as a bar plot or a pairwise plot.

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.

Protein 2 and 3 are most similar in their B-factor trends. We could quantify this by doing the following code:

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



This essentially clusters the proteins based on their similarity, hence it gives a quantification of which proteins are most similar to one another.

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

```
#analyze_kinase <- function(kinase){
 # s <- read.pdb(kinase)
 # s.chainA <- trim.pdb(s, chain="A", elety="CA")
 # s.b <- s.chainA$atom$b
 # plotb3(s.b, sse=s.chainA, typ="l", ylab="Bfactor")
#}

#kinases <- list(___)
#for (residue in kinases){
 # analyze_kinase(residue)
#}
```