class06_hw

Anna Waters (PID: A16271985)

2024-01-28

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$a))
df$a
```

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

```
df$b <- (df$b - min(df$a)) / (max(df$b) - min(df$b)) #miss type of max()
df$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)) #miss type of max()
```

Above code block with misstyping fixed

```
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$a))
df$a
```

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

```
df$b <- (df$b - min(df$b)) / (max(df$b) - min(df$b)) #miss type of max() fixed
df$c <- (df$c - min(df$c)) / (max(df$c) - min(df$c))
df$d <- (df$d - min(df$d)) / (max(df$d) - min(df$d)) #miss type of max() fixed
```

Above code is subtracting every cell in the column by the minimum value then dividing by the max value subtracted by the minimum and replacing the column which results in normalized data

```
df$a <- (df$a - min(df$a)) / (max(df$a) - min(df$a))
```

Im going to switch out the variable for something generalizable

```
x <- df$a
#using min() to find the minimum value & max() to find the max value to normalize the data
x <- (x -min(x)) / (max(x)-min(x))

x
```

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

Looks like the same answer as the original so I am going forward with making it into a function.

```
#name: norm_fun
# creating the function by calling `function()`
#input: x = column of the data frame
norm_fun <- function(x) {
  #input that uses min() and max() to normalize the input
  x <- (x - min(x)) / (max(x)-min(x))
  #passing the value of x outside the function
  return(x)
}
```

Testing on the df

```
df_test <- data.frame(a=1:10, b=seq(200,400,length=10),c=11:20,d=NA)
#using apply function to test on the data frame across every column (margin=2)
apply(df_test,MARGIN = 2,FUN = norm_fun)
```

```
      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
```

Function test looks correct! Looks like the above function works!

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.

I installed `bio3d()` using the consol below with `install.packages()` and will library it in the chunck.

```
# Can you improve this analysis code?
# pulling bio3d package into the workspace to use
library(bio3d)
# `read.pdb()` reads PDB files and returns a large pdb object
s1 <- read.pdb("4AKE") # kinase with drug
```

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

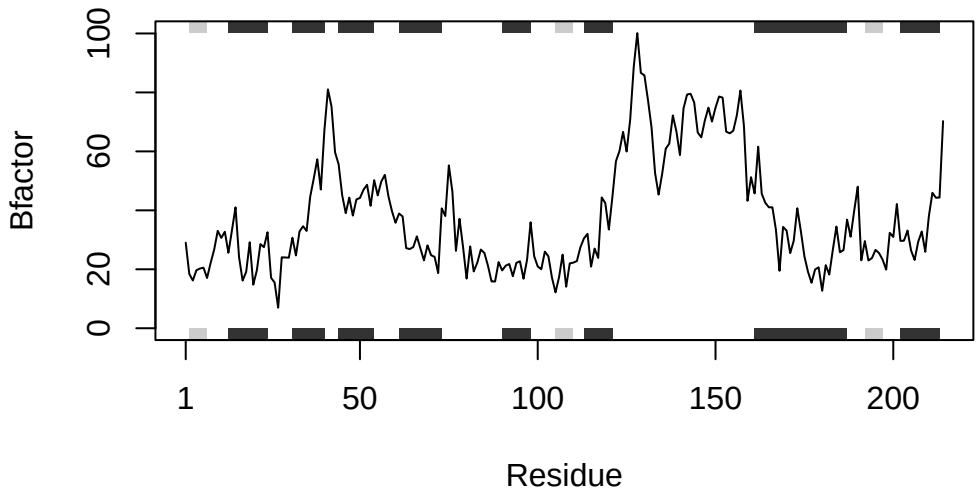
```
s2 <- read.pdb("1AKE") # kinase no drug
```

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

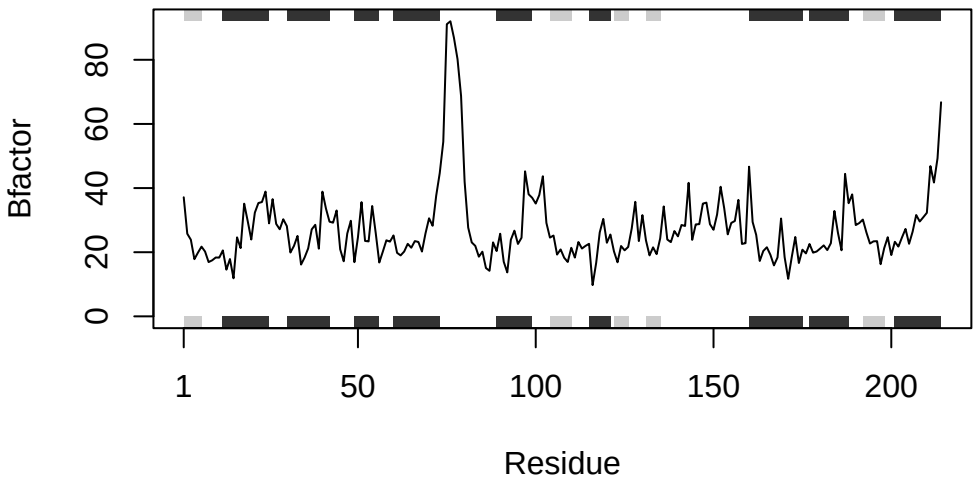
```
s3 <- read.pdb("1E4Y") # kinase with drug
```

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

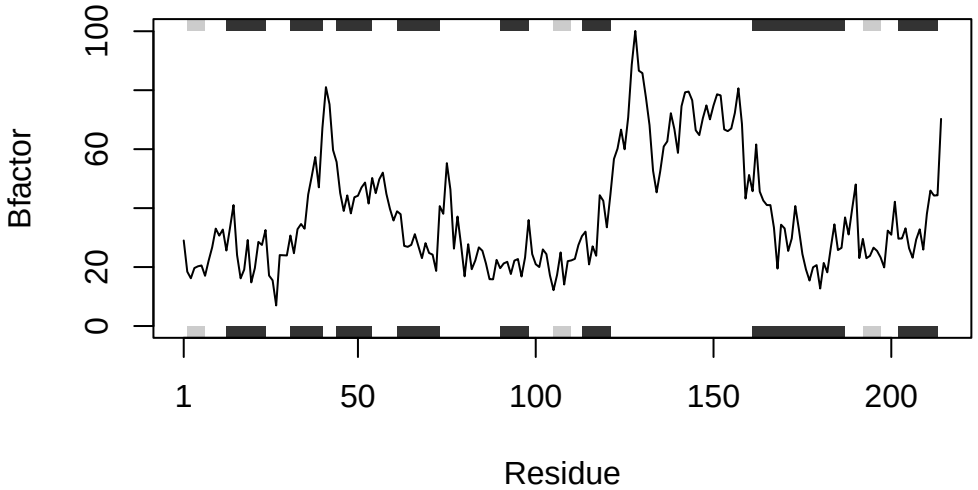
```
#trim.pdb creates a smaller pdb by subsetting the type of chain (A) and the type of atom (
s1.chainA <- trim.pdb(s1, chain="A", elety="CA")
s2.chainA <- trim.pdb(s2, chain="A", elety="CA")
s3.chainA <- trim.pdb(s1, chain="A", elety="CA")
#setting a variable to equal the b atom
s1.b <- s1.chainA$atom$b
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?

```
s1
```

```
 Call:  read.pdb(file = "4AKE")

   Total Models#: 1
     Total Atoms#: 3459,  XYZs#: 10377  Chains#: 2  (values: A B)

     Protein Atoms#: 3312  (residues/Calpha atoms#: 428)
     Nucleic acid Atoms#: 0  (residues/phosphate atoms#: 0)

     Non-protein/nucleic Atoms#: 147  (residues: 147)
     Non-protein/nucleic resid values: [ HOH (147) ]

   Protein sequence:
      MRIILLGAPGAGKGTQAQFIMEKYGIPQISTGDMLRAAVKSGSELGKQAKDIMDAGKLVT
      DELVIALVKERIAQEDCRNGFLLDGFPRTIPQADAMKEAGINVDYVLEFDVPDELIVDRI
      VGRRVHAPSGRVYHVKFNPPKVEGKDDVTGEELTTRKDDQEETVRKRLVEYHQMTAPLIG
      YYSKEAEAGNTKYAKVDGTKPVAEVRADLEKILGMRIILLGAPGA...<cut>...KILG

+ attr: atom, xyz, seqres, helix, sheet,
        calpha, remark, call
```

```
typeof(s1)
```

```
[1] "list"
```

```
?read.pdb
```

`Read.pbd()` returns a PDB object in the form of a list.

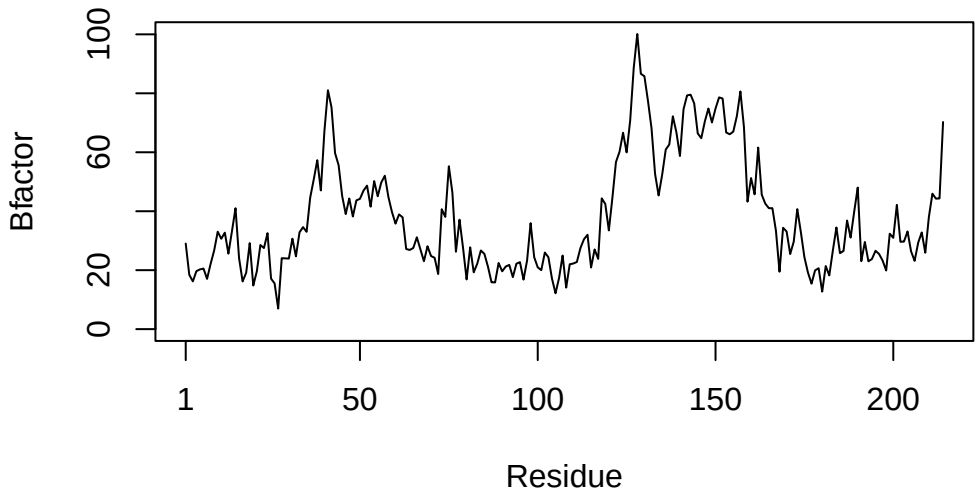
Q2. What does the trim.pdb() function do?

```
?trim.pdb
```

`trim.pdb()` creates a small pdb object by subsetting the larger pdb. The chain parameter and elety specify which chain and which type of atom is being subset into the new 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?

```
#removing the sse input of the plot removes the black and grey rectangles which were pulli
plotb3(s1.b, typ="l", ylab="Bfactor")
```



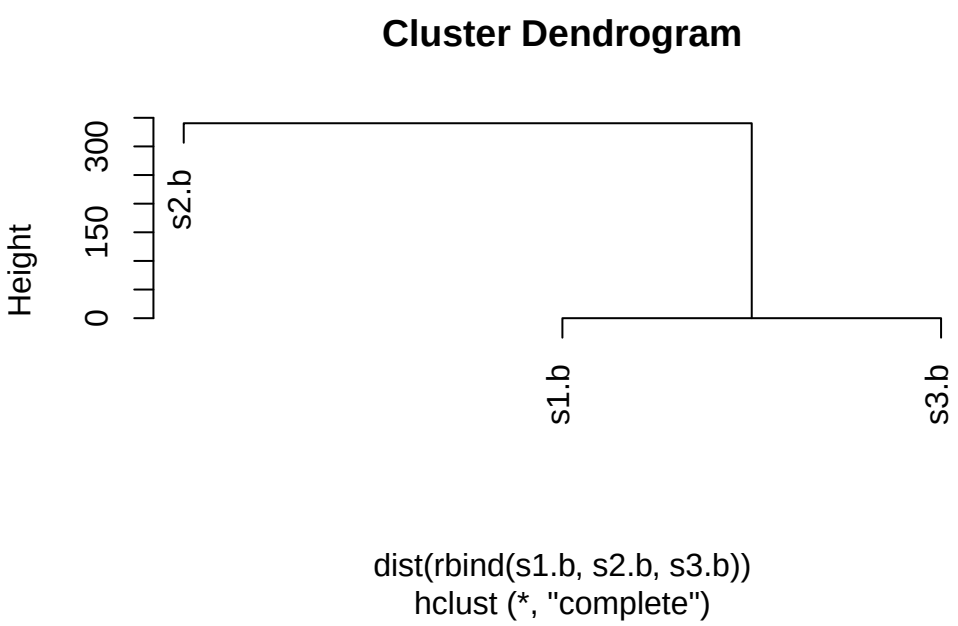
In this case, by removing the SSE component of the plot, the annotation information about where Chain A relative to the protein residue is removed from the plot.

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

A better plot to compare the different proteins would be a dendrogram or alignment plot because it would better show their similarities in structure.

Q5. Which proteins are more similar to each other in their B-factor trends. How could you quantify this?

```
hc <- hclust( dist( rbind(s1.b, s2.b, s3.b) ) )
#rbind creates a row with the three inputs
#dist() creates a distance matrix
#hclust() performs hierarchal clusteirng on the distance matrix to create the dendrogram
plot(hc)
```



Based on the dendrogram, proteins 1 and 3 are the most similar to each other.

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

First, I am going to simplify the code by annotating what each step of the processes is doing.

```
# `read.pdb()` reads PDB files and returns a large pdb object
s1 <- read.pdb("4AKE") # kinase with drug
```

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

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
/var/folders/7b/5xrlz_vx2c5d__w683y_p5pm0000gn/T//RtmptwLB5R/4AKE.pdb exists.
Skipping download
```

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

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

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
/var/folders/7b/5xrlz_vx2c5d__w683y_p5pm0000gn/T//RtmptwLB5R/1AKE.pdb exists.
Skipping download
```

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

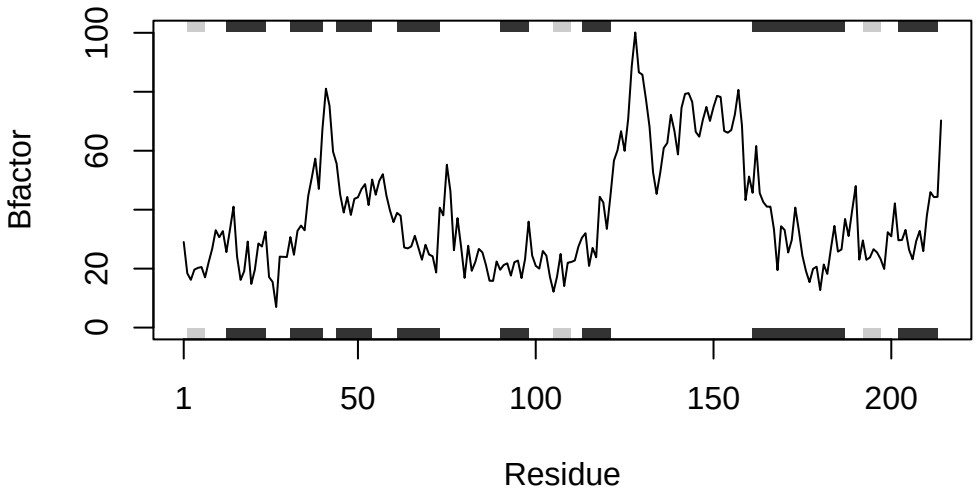
```
s3 <- read.pdb("1E4Y") # kinase with drug
```

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

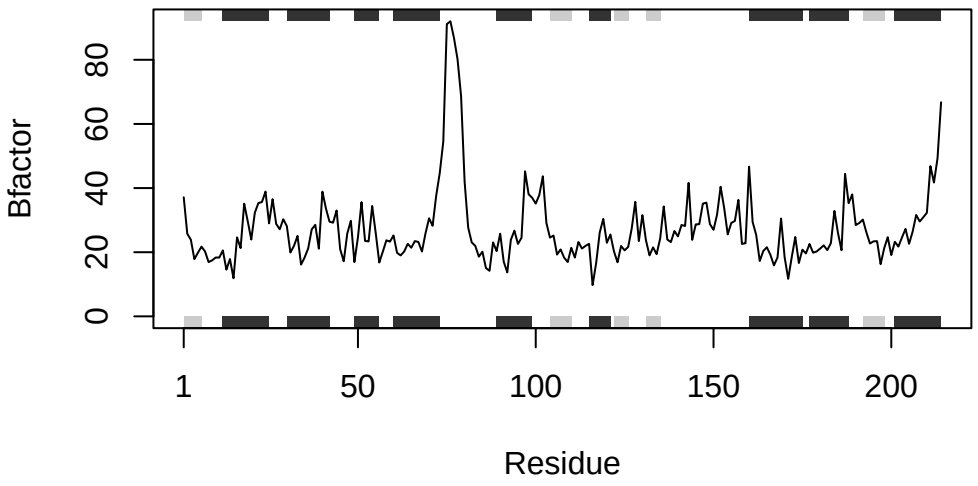
```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
/var/folders/7b/5xrlz_vx2c5d__w683y_p5pm0000gn/T//RtmptwLB5R/1E4Y.pdb exists.
Skipping download
```

```
#trim.pdb creates a smaller pdb by subsetting the type of chain (A) and the type of atom (
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")
#setting a variable to equal the b atom
```

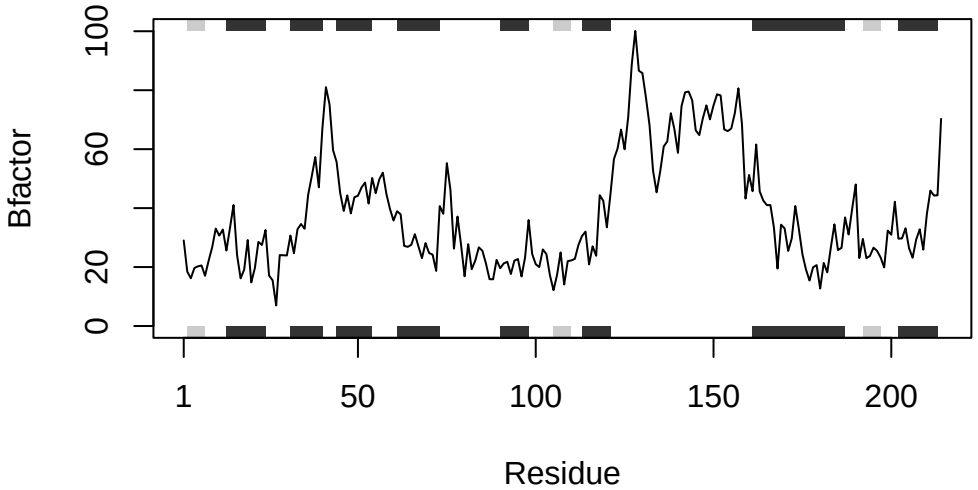
```
s1.b <- s1.chainA$atom$b
s2.b <- s2.chainA$atom$b
s3.b <- s3.chainA$atom$b
# Plotting a line graph (typ = "l") of residues (s1.b) vs Bfactor with the chain A annotat
plotb3(s1.b, sse=s1.chainA, typ="l", ylab="Bfactor")
```




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




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



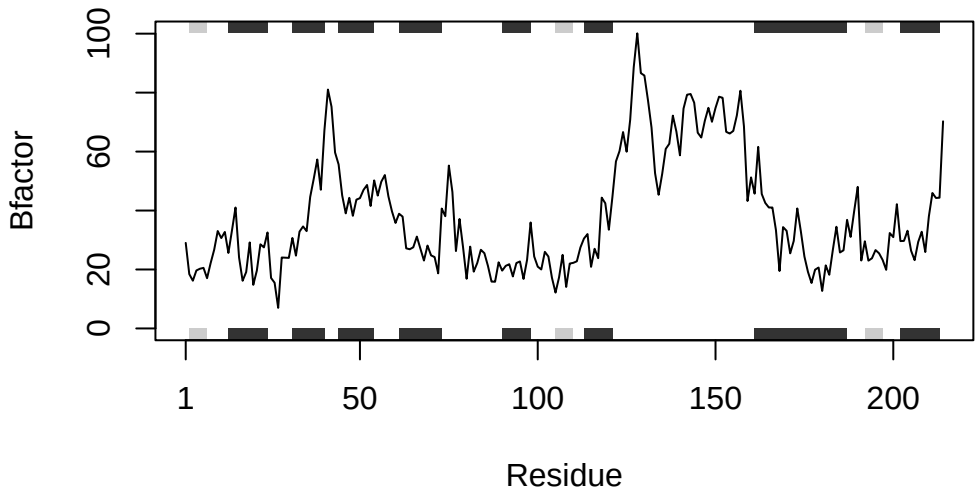
Now I am going to make a simplified snippet for just one of the protein sequences.

```
#read in the pdb with input between the ""
s1 <- read.pdb("4AKE") # kinase with drug
```

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

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
/var/folders/7b/5xrlz_vx2c5d__w683y_p5pm0000gn/T//RtmptwLB5R/4AKE.pdb exists.
Skipping download
```

```
#subset the read pdb for chain A and the atom type
s1.chainA <- trim.pdb(s1, chain="A", elety="CA")
#isolate atom type b from chain A in the pdb and assign it to a new vector
s1.b <- s1.chainA$atom$b
#plot the b factor by residue with chain A annotations
plotb3(s1.b, sse=s1.chainA, typ="l", ylab="Bfactor")
```



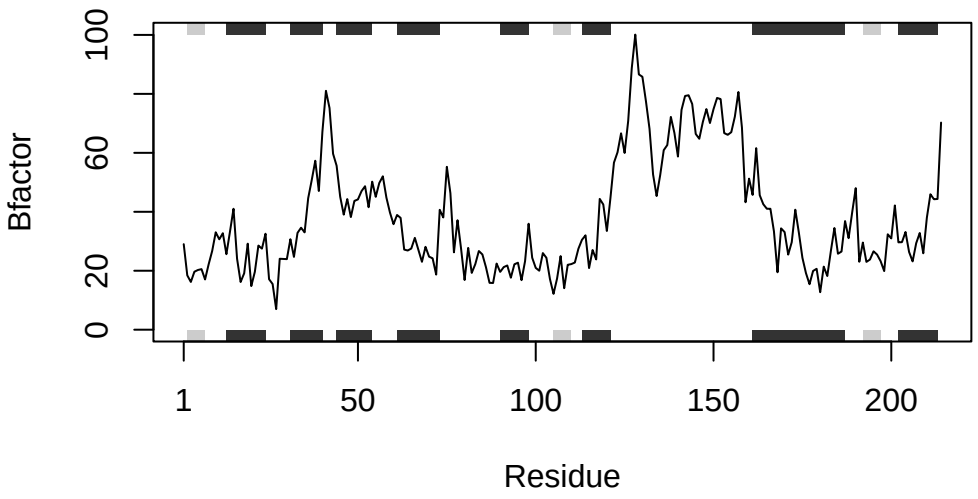
I am going to further simplify the snippet by replacing the specific variable with a general one and test it to see the answer is the same. I am setting the read in variable as a generic.

```
# setting the read in PDB as a generic variable

#read in the pdb with input between the ""
p <- read.pdb("4AKE") # kinase with drug
```

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

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
/var/folders/7b/5xrlz_vx2c5d__w683y_p5pm0000gn/T//RtmptwLB5R/4AKE.pdb exists.
Skipping download
```

```
#subset the read pdb for chain A and the atom type
p.chainA <- trim.pdb(p, chain="A", elety="CA")
#isolate atom type b from chain A in the pdb and assign it to a new vector
p.b <- p.chainA$atom$b
#plot the b factor by residue with chain A annotations
plotb3(p.b, sse=p.chainA, typ="l", ylab="Bfactor")
```



The results of the generalized is the same so I am going to start forming the function body and inputs off of the simplified snippet.

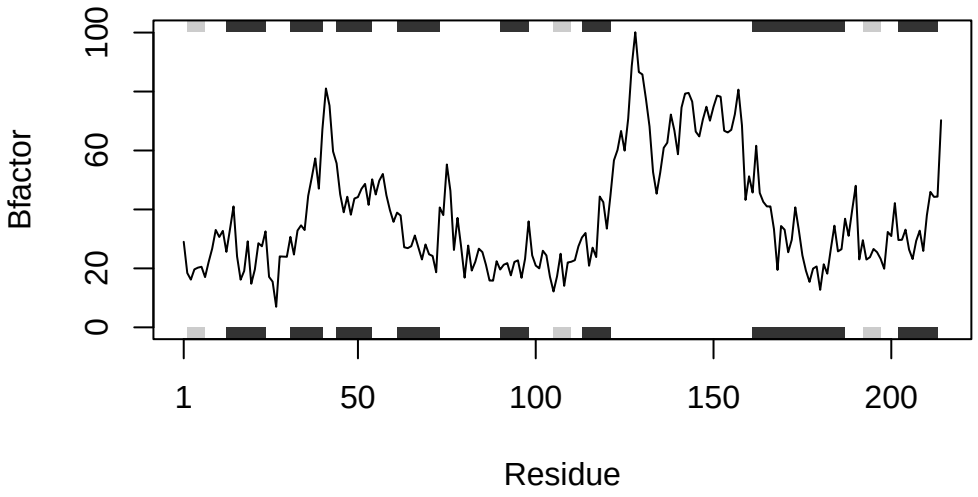
```
#Requires `plot3b()`
#Inputs:
#p = PDB name to read in, needs "" ex: "4AKE"
#c = which chain to subset (chain =), needs "", assumes chain "A" if no input
#e = type of Atom to subset (elety =), needs "", assumes "CA" if no input
#Function creates B factor plot of Chain A and Atom type CA


Protein_plot <- function(p,c ="A", e = "CA"){

  #read in the pdb with input from p
protein <- read.pdb(p)

#subset the read pdb for chain A and the atom type unless specified
protein.chainA <- trim.pdb(protein, chain = c , elety = e)

#isolate atom type b from chain A in the pdb and assign it to a new vector
protein.b <- protein.chainA$atom$b

#plot the b factor by residue with chain A annotations
Bfactor_plot <- plotb3(protein.b, sse=protein.chainA, typ="l", ylab="Bfactor")

#Pulls the plot from the function and adds it to the global environment for other use
return(Bfactor_plot)


}
```

Testing `Protein_plot()` on the previous test data to ensure the function is working correctly.

```
Protein_plot("4AKE")
```

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

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
/var/folders/7b/5xrlz_vx2c5d__w683y_p5pm0000gn/T//RtmptwLB5R/4AKE.pdb exists.
Skipping download
```



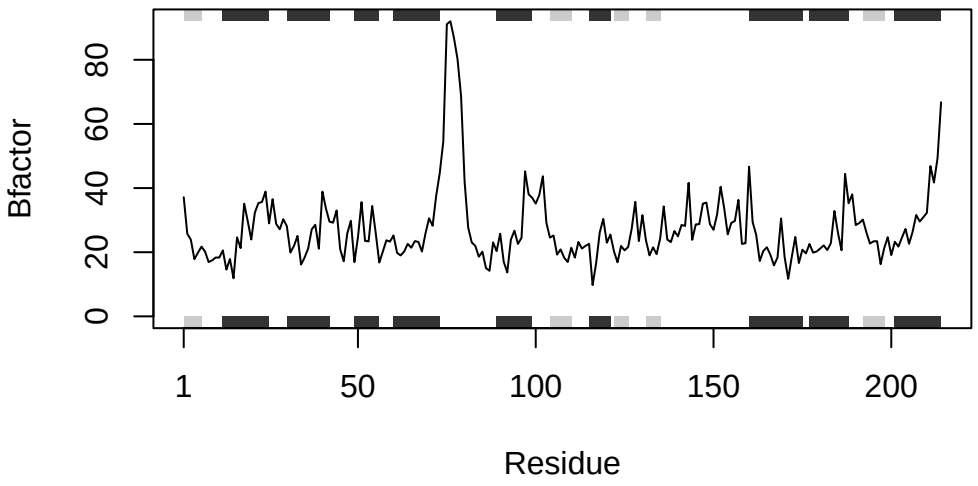

```
NULL
```

```
Protein_plot("1AKE")
```

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

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
/var/folders/7b/5xrlz_vx2c5d__w683y_p5pm0000gn/T//RtmptwLB5R/1AKE.pdb exists.
Skipping download
```

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

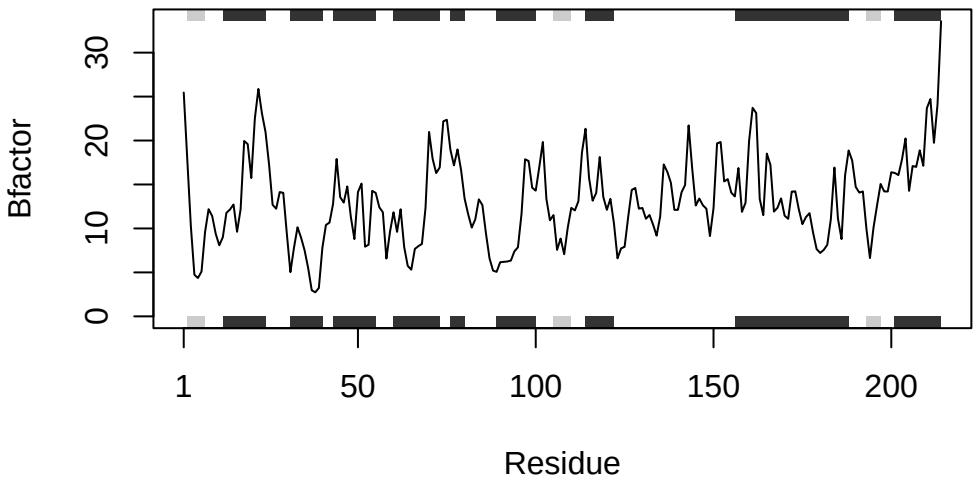



```
NULL
```

```
Protein_plot("1E4Y")
```

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

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):
/var/folders/7b/5xrlz_vx2c5d__w683y_p5pm0000gn/T//RtmptwLB5R/1E4Y.pdb exists.
Skipping download
```




```
NULL
```

Success! Three plots that plot the B factor of each protein.

The function works to create a B-factor vs residue plot for each protein using the `read.pdb()`, `trim.pdb()` and `bio3d()`. The function uses read.pdb to read in the protein and subsets chain A using `trim.pdb`. Then `bio3d()` plot3d function is used to plot the b factor vs the residue with the chain A annotation.

The function name is: **Protein_plot**

There are three inputs, 1 required and 2 with preset options:

-p = the pdb name to read in surrounded with " ", required: this is used in the `read.pdb()` step

-c = the chain identifier surrounded with "", preset to"A" :required for subsetting and used as the value in `trim.pdb()` for the chain argument

-e = the atom type identifier surrounded with "", preset to"CA": required for subsetting and used as the value in `trim.pdb()` for elety argument