lab_6_supplement Writing Functions

Kelly_F

10/20/2021

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) #create df
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))

# Assuming that the code is meant to use the same input $column for all min & max functions per line ab
mask <- data.frame(a=1:10, b=seq(200,400,length=10),c=11:20,d=NA)

# Write the function. Function makes all columns of dataframe the same.
df_analysis <- function(column){
  column <- (column - min(column)) / (max(column) - min(column))
}

# Test function on one column of data frame
print(df_analysis(mask$a))
```

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

```
# Apply function to entire data frame
analyzed_mask <- apply(mask, 2, df_analysis)
analyzed_mask
```

```
##              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.

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.

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

Q2. What does the trim.pdb() function do? The trim.pdb() function produces a new PDB object that contains a subset of atoms from a 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? bot=FALSE. They represent the residues of secondary structure object, which in this case represents alpha and beta strands.

Q4. What would be a better plot to compare across the different proteins? Stack graphs across different proteins or plot all proteins in the same graph

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

Required Homework

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

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

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

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

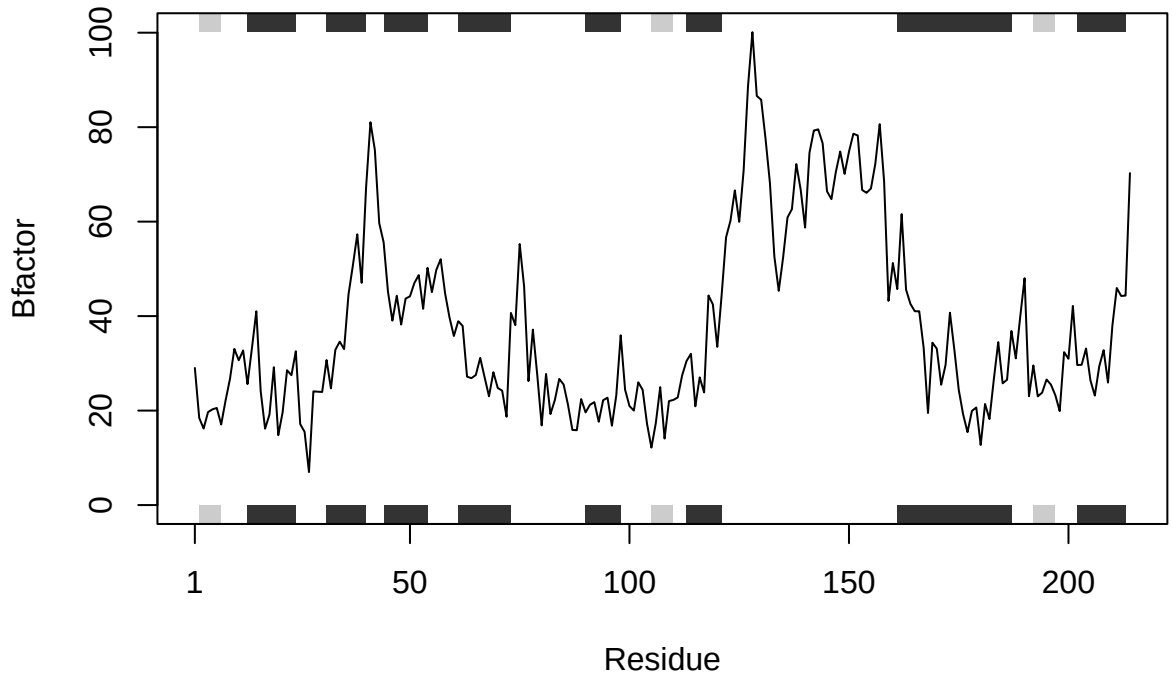
```
s2 <- read.pdb("1AKE")  # kinase no drug
```

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

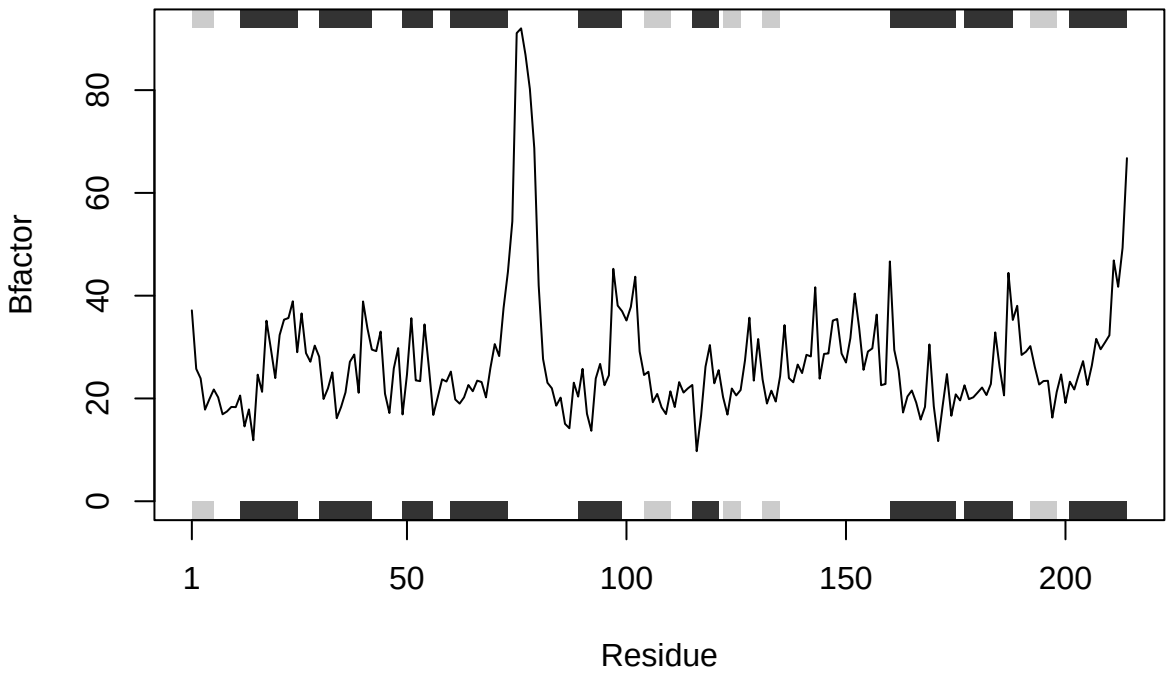
```
s3 <- read.pdb("1E4Y")  # kinase with drug
```

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

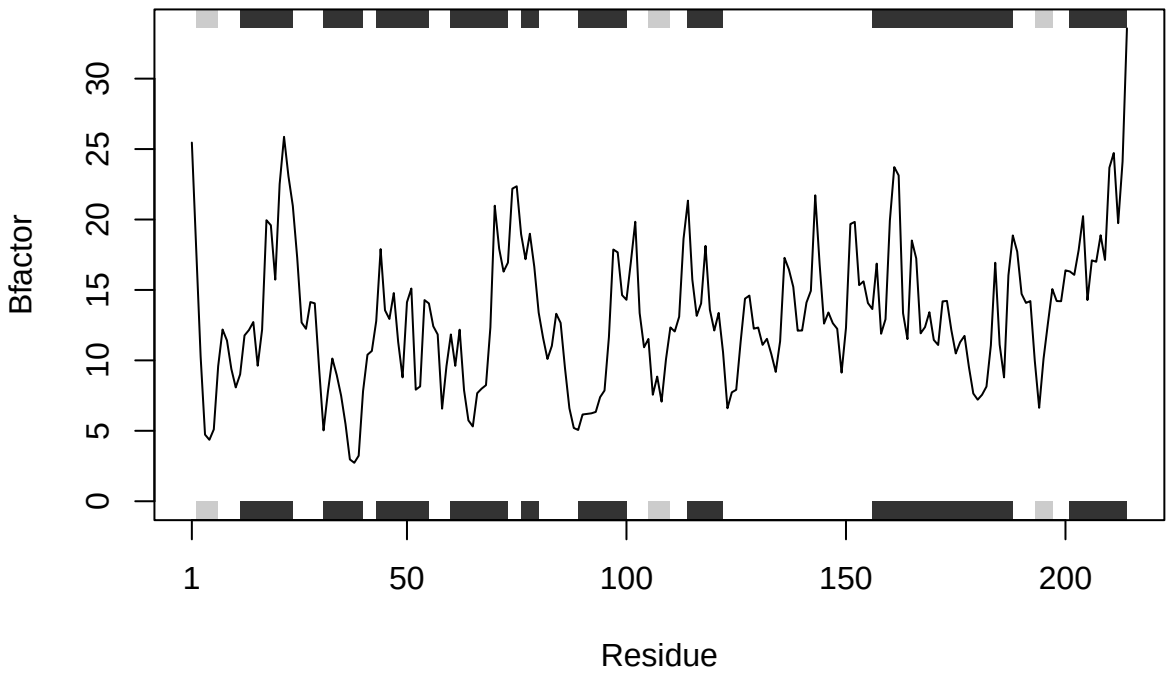
```
s1.chainA <- trim.pdb(s1, chain="A", elety="CA")
s2.chainA <- trim.pdb(s2, chain="A", elety="CA")
s3.chainA <- trim.pdb(s3, chain="A", elety="CA") #typo fixed, s1 -> s3

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")
```




```
#1 Type of object returned from read.pdb
typeof(s1)
```

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

```
#2 Determine what trim.pdb function does
?trim.pdb

# Create function to simplify code from above
plot_Bfactor <- function(PDB_code){ #input PDB protein code
  protein <-read.pdb(PDB_code) #load in protein PDB object
  protein.chainA <- trim.pdb(protein, chain="A", elety="CA") #trim object to subset of atoms of interes
  protein.b <- protein.chainA$atom$b #assign B factor value to variable for plotting
  plotb3(protein.b, sse=protein.chainA, typ="l", ylab="Bfactor", bot=FALSE) #plot Bfactor value vs resi
}

# See if function works on single protein
plot_Bfactor("4AKE")
```

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

```
## Warning in get.pdb(file, path = tempdir(), verbose = FALSE): /var/folders/9g/
## k400z9fd313fgqsww52j2zn40000gp/T//RtmpsO34bZ/4AKE.pdb exists. Skipping download
```

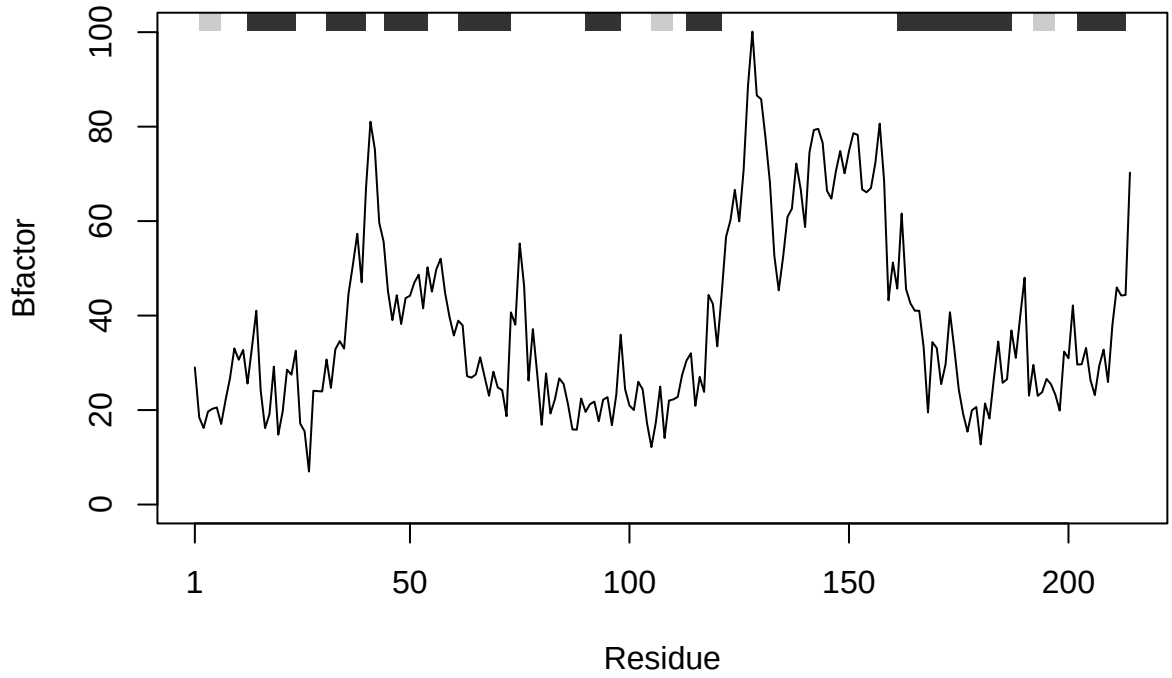
```
# Create list of all proteins you want to run through function
proteins <- c("4AKE", "1AKE", "1E4Y")

# Apply function to all proteins of interest
lapply(proteins, plot_Bfactor)
```

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

```
## Warning in get.pdb(file, path = tempdir(), verbose = FALSE): /var/folders/9g/
## k400z9fd3l3fgqsww52j2zn40000gp/T//RtmpsO34bZ/4AKE.pdb exists. Skipping download
```

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

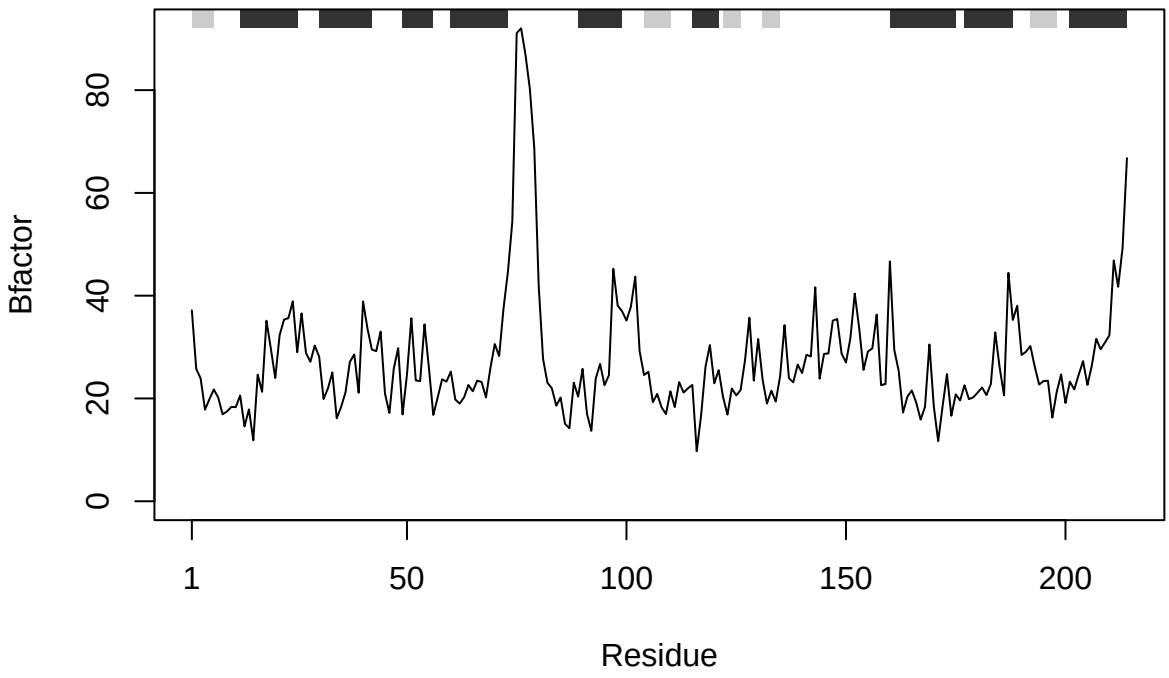
```
## Warning in get.pdb(file, path = tempdir(), verbose = FALSE): /var/folders/9g/
## k400z9fd3l3fgqsww52j2zn40000gp/T//RtmpsO34bZ/1AKE.pdb exists. Skipping download
```

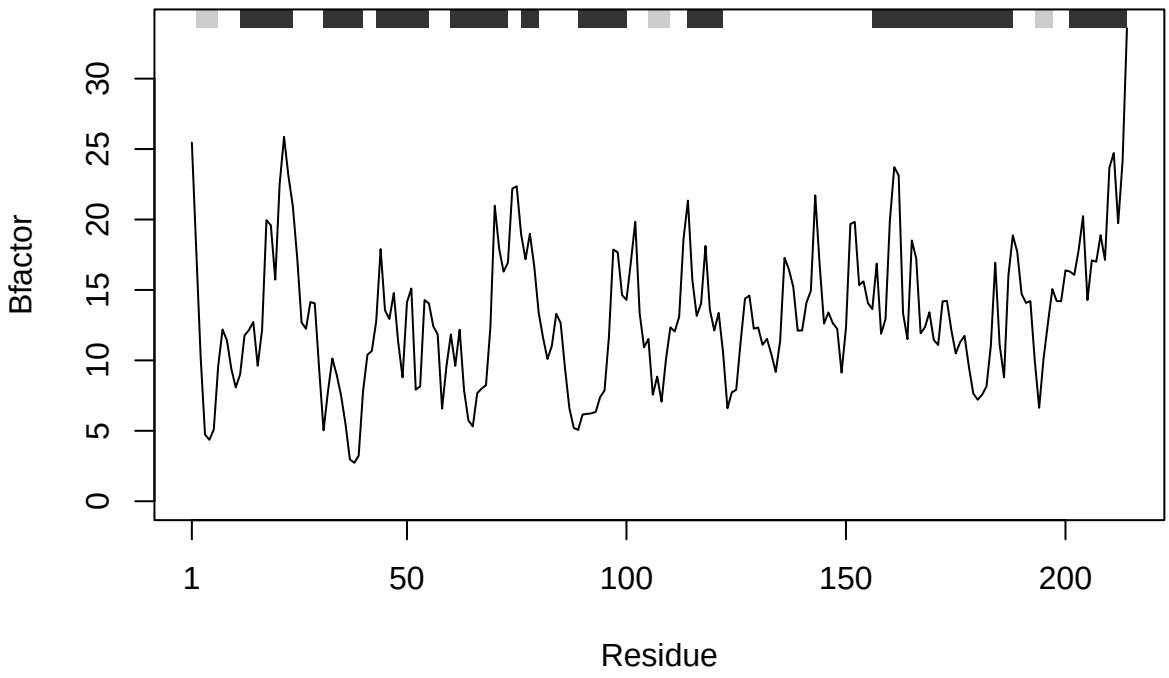



```
##    PDB has ALT records, taking A only, rm.alt=TRUE
```

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

```
## Warning in get.pdb(file, path = tempdir(), verbose = FALSE): /var/folders/9g/
## k400z9fd3l3fgqsww52j2zn40000gp/T//RtmpsO34bZ/1E4Y.pdb exists. Skipping download
```


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




```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
```