Class 6: R Functions

Christopher Cheng

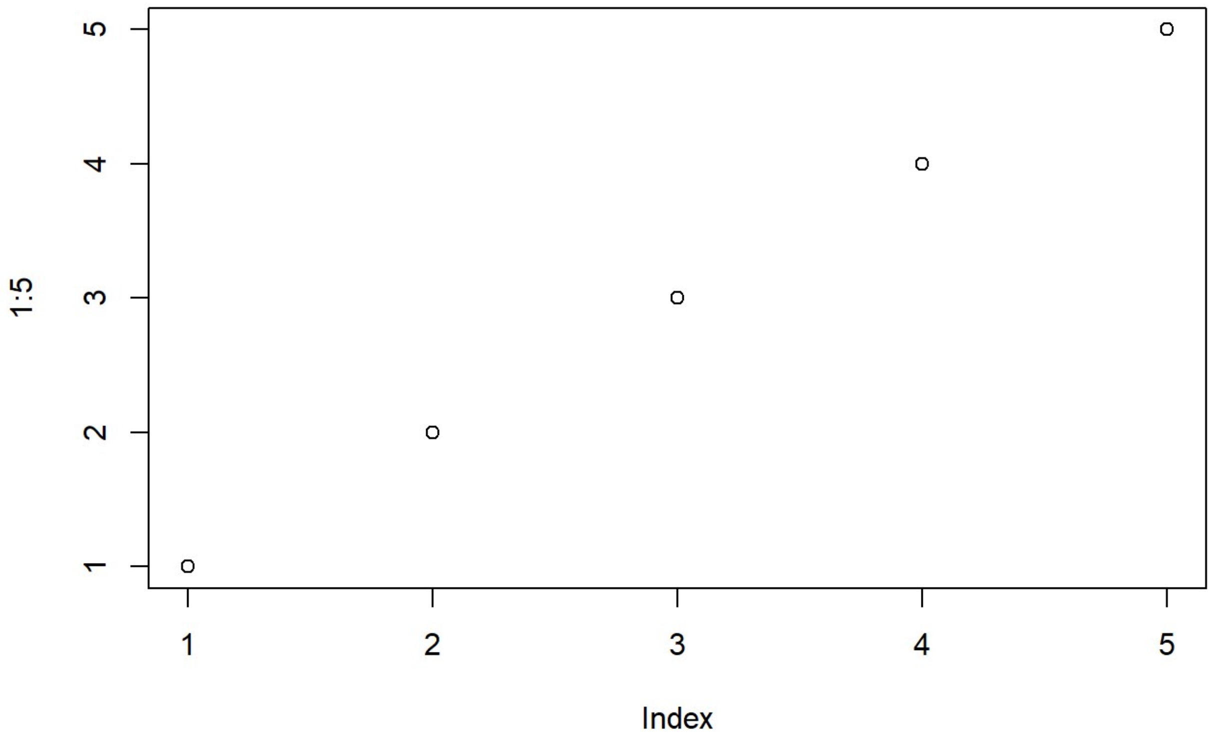
01/23/2020

Functions

Level 2 Heading

Level 3 Heading

```
#this is a silly plot

plot(1:5)
```



Let's see more about **file import** (i.e reading files into R). The main read function in base R is `read.table()`

We need to add arguments to get this file imported

```
t1 <- read.table("test1.txt", sep = ",", header = TRUE)
```

Or I could just use `read.csv()` which has the arguments I want in this case!

```
t1 <- read.csv("test1.txt")
t1
```

```
##   Col1 Col2 Col3
## 1    1    2    3
## 2    4    5    6
## 3    7    8    9
## 4    a    b    c
```

For the last two tables, I'll use whatever **I WANT**

```
t2 <- read.table("test2.txt", header = TRUE, sep = "$")
t3 <- read.table("test3.txt", header = FALSE, sep = "")
```

Back to functions

Our first example function:

```
add <- function(x, y = 1){
  # Sum the input x and y
  x + y
}
```

Let's try using this function

```
add(c(1,2,3), c(1,2,3))
```

```
## [1] 2 4 6
```

This is our second function

```
rescale <- function(x){
  rng <- range(x)
  (x-rng[1]) / (rng[2] - rng[1])
}
```

```
# Test on a small example where you know the answer
rescale(1:10)
```

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

```
# How would you get your function to work here…
rescale( c(1,2,NA,3,10) )
```

```
## [1] NA NA NA NA NA
```

```
x <- c(1,2,NA,3,10)
rng <- range(x)
rng
```

```
## [1] NA NA
```

```
x <- c(1,2,NA,3,10)
rng <- range(x, na.rm = TRUE)
rng
```

```
## [1]  1 10
```

```
rescale2 <- function(x){
  rng <- range(x, na.rm = TRUE)
  (x-rng[1]) / (rng[2] - rng[1])
}
```

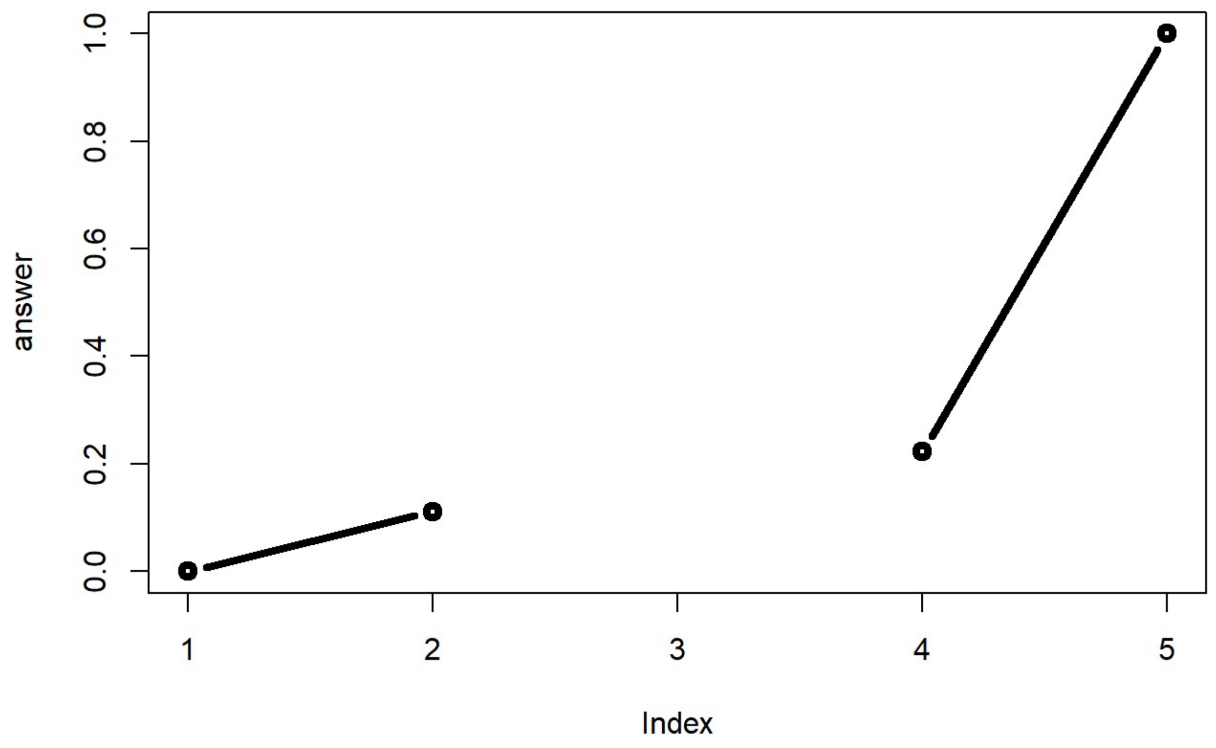
```
rescale2(c(1,2,NA,3,10))
```

```
## [1] 0.0000000 0.1111111        NA 0.2222222 1.0000000
```

```r
rescale3 <- function(x, na.rm=TRUE, plot=FALSE) {
  rng <-range(x, na.rm=na.rm)
  print("Hello")

  answer <- (x - rng[1]) / (rng[2] - rng[1])

  print("is it me you are looking for?")

  if(plot) {
    print("Don't sing again please!")
    plot(answer, typ="b", lwd=4)
  }

  print("I can see it in ...")

  return(answer)
}
```

```r
rescale3(x, plot = TRUE)
```

```
## [1] "Hello"
## [1] "is it me you are looking for?"
## [1] "Don't sing again please!"
```



```
## [1] "I can see it in ..."
```

```
## [1] 0.0000000 0.1111111        NA 0.2222222 1.0000000
```

Moving to Section B of Lab Workup

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

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

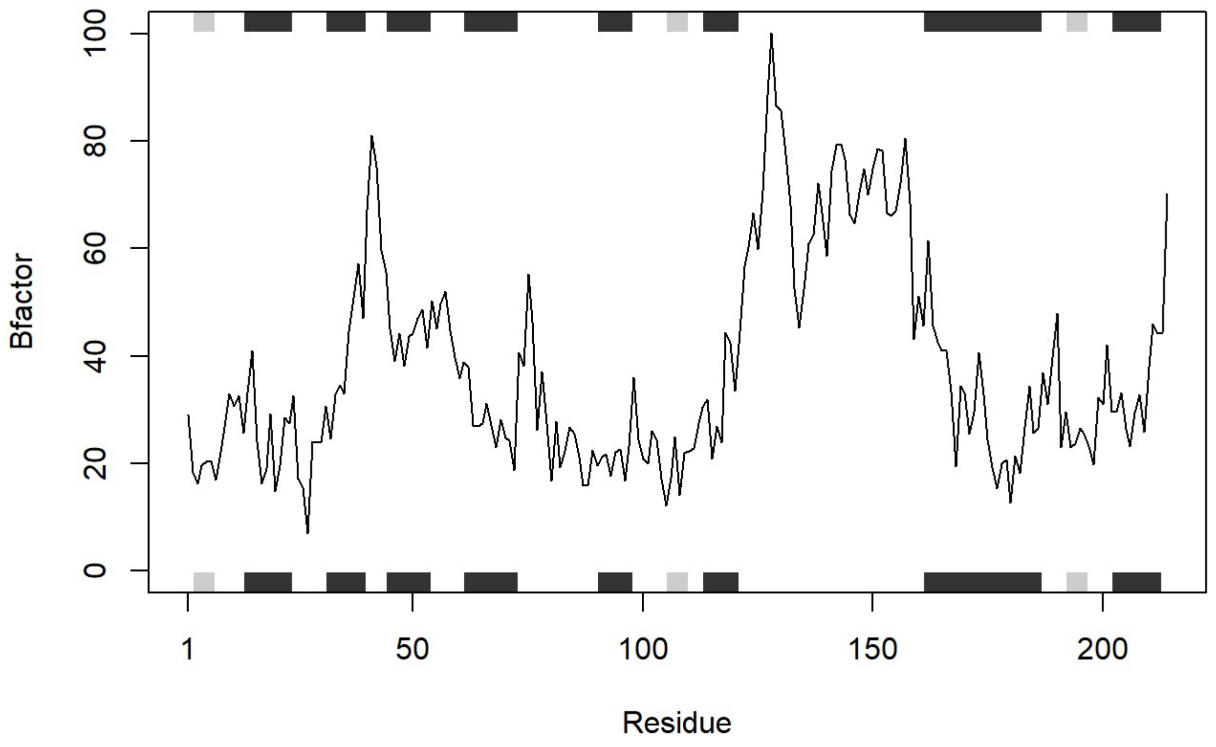
```
s2 <- read.pdb("1AKE") # kinase no drug
```

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

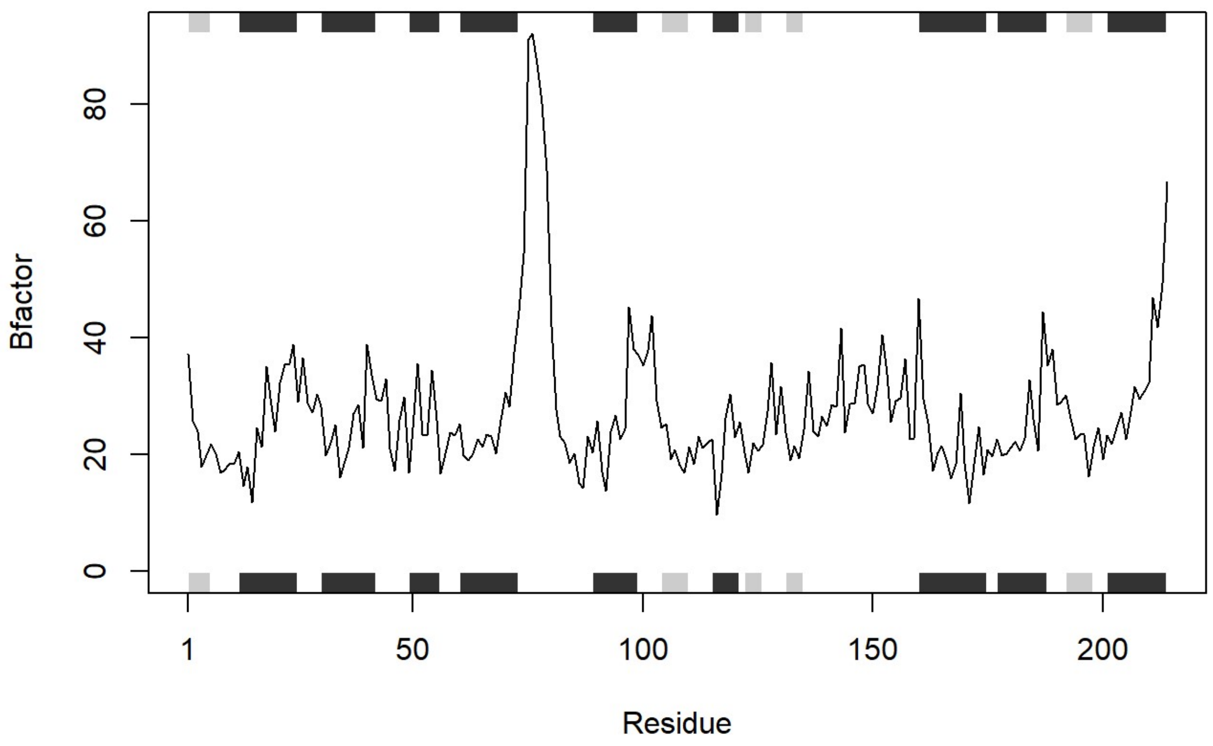
```
s3 <- read.pdb("1E4Y") # kinase with drug
```

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

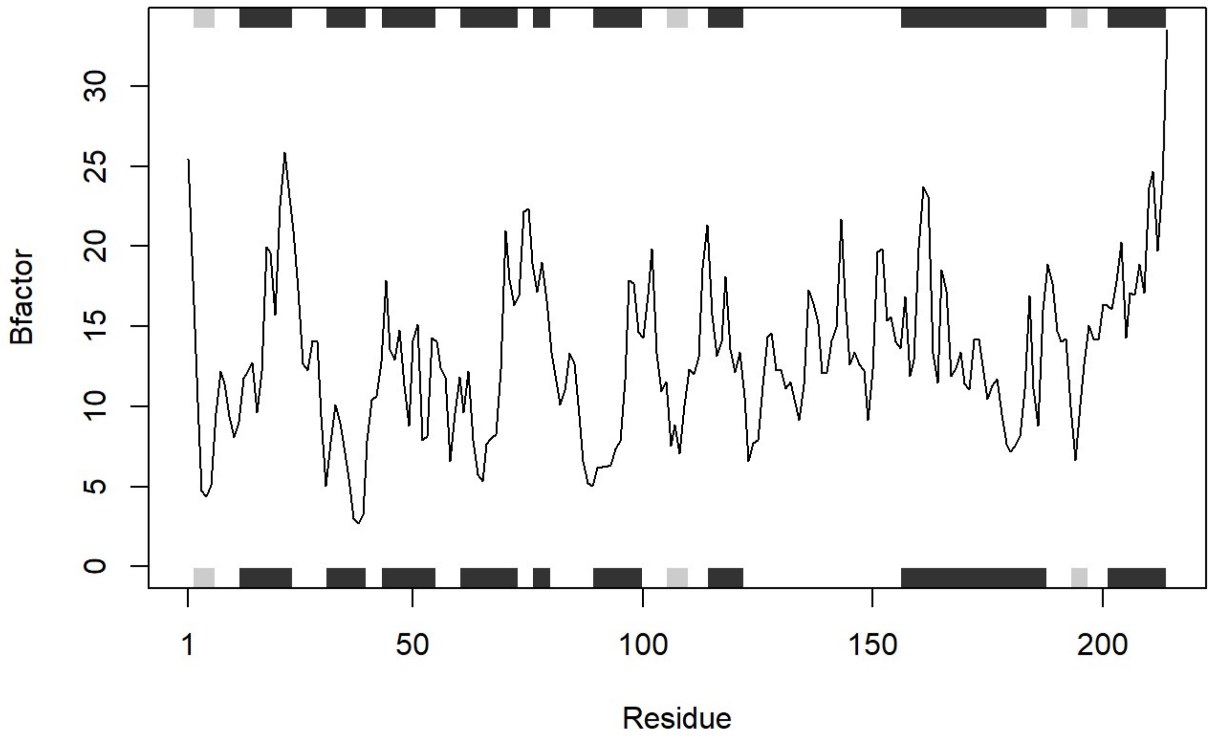
```
s1.chainA <- trim.pdb(s1, chain="A", elety="CA")
s2.chainA <- trim.pdb(s2, chain="A", elety="CA")
s3.chainA <- trim.pdb(s3, chain="A", elety="CA")

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?

It is a list of 8 things

```
class(s1)
```

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

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

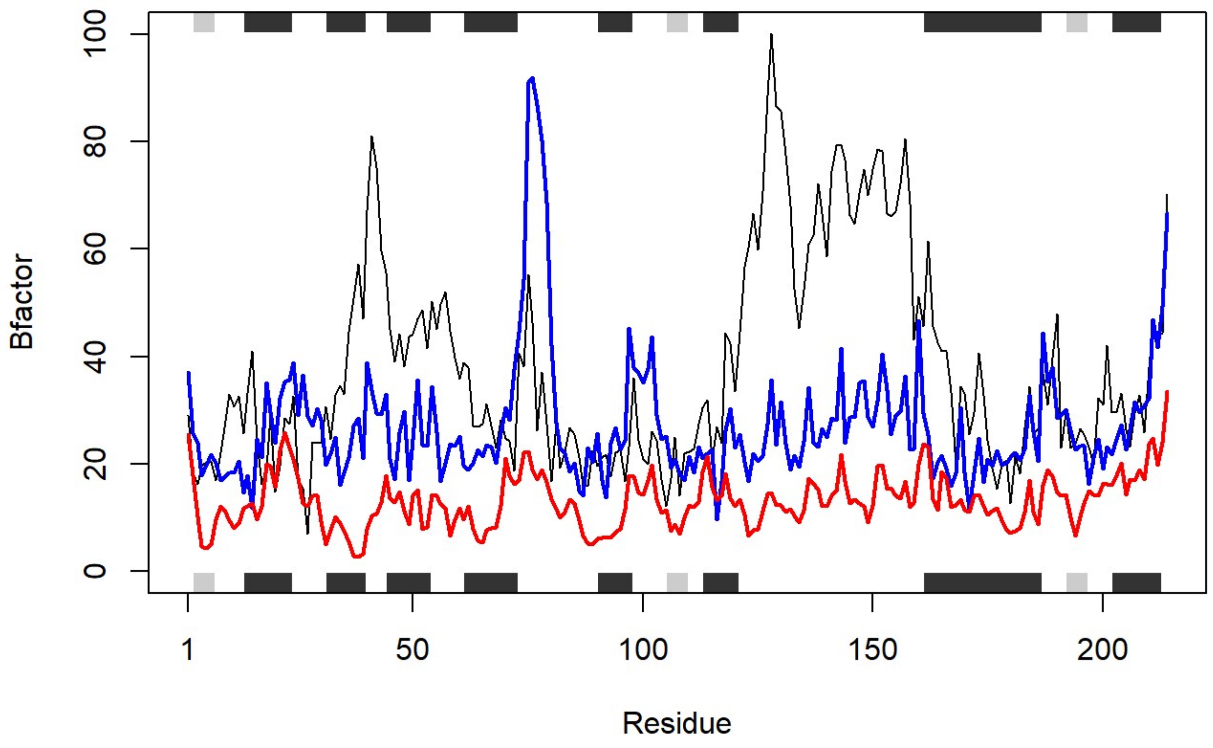
Producs a new smaller PDB object containing a subset of atoms. Trims it down

```
?trim.pdb
```

```
## starting httpd help server ... done
```

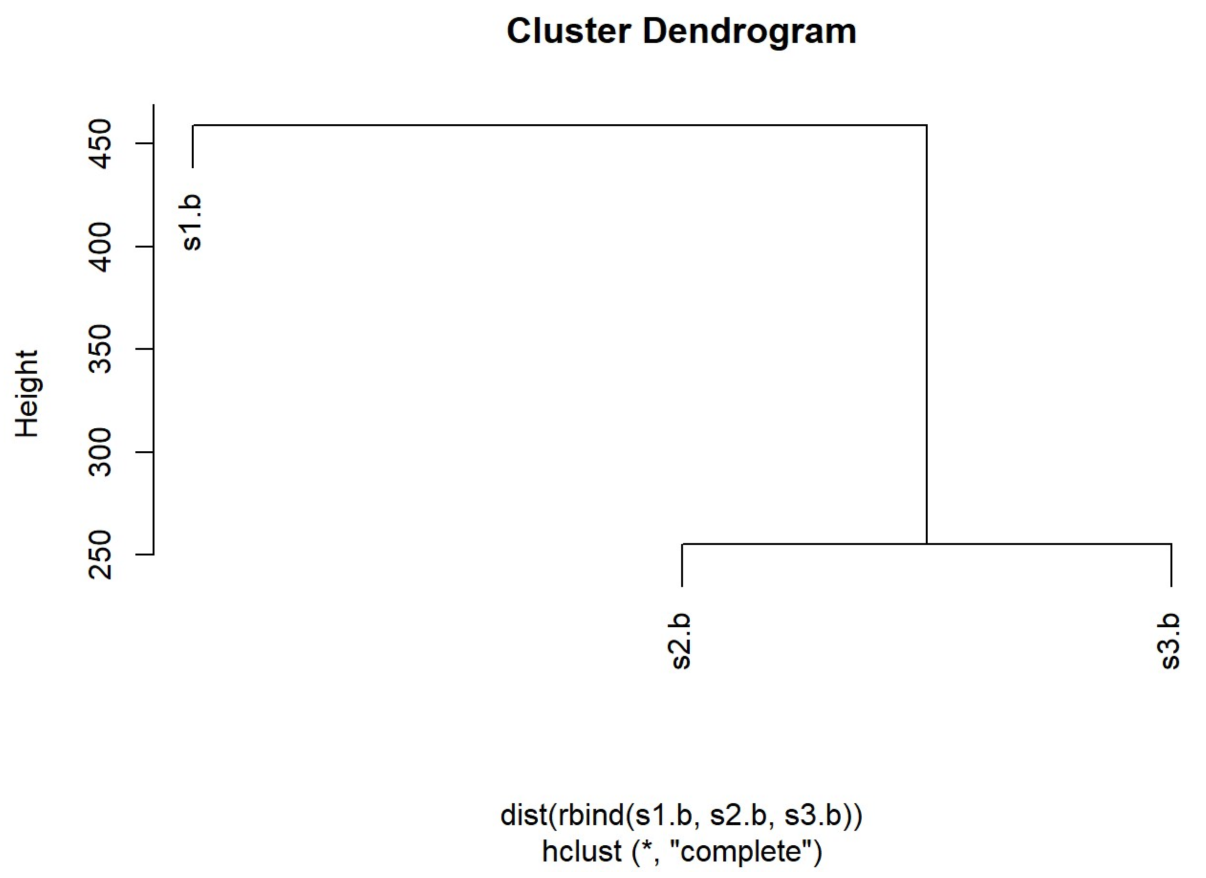
Q4. What would be a better plot to compare across the different proteins?

```
plotb3(s1.b, sse=s1.chainA, typ="l", ylab="Bfactor")
points(s2.b, typ = "l", col = "blue", lwd = 2)
points(s3.b, typ = "l", col = "red", lwd = 2)
```



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.

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



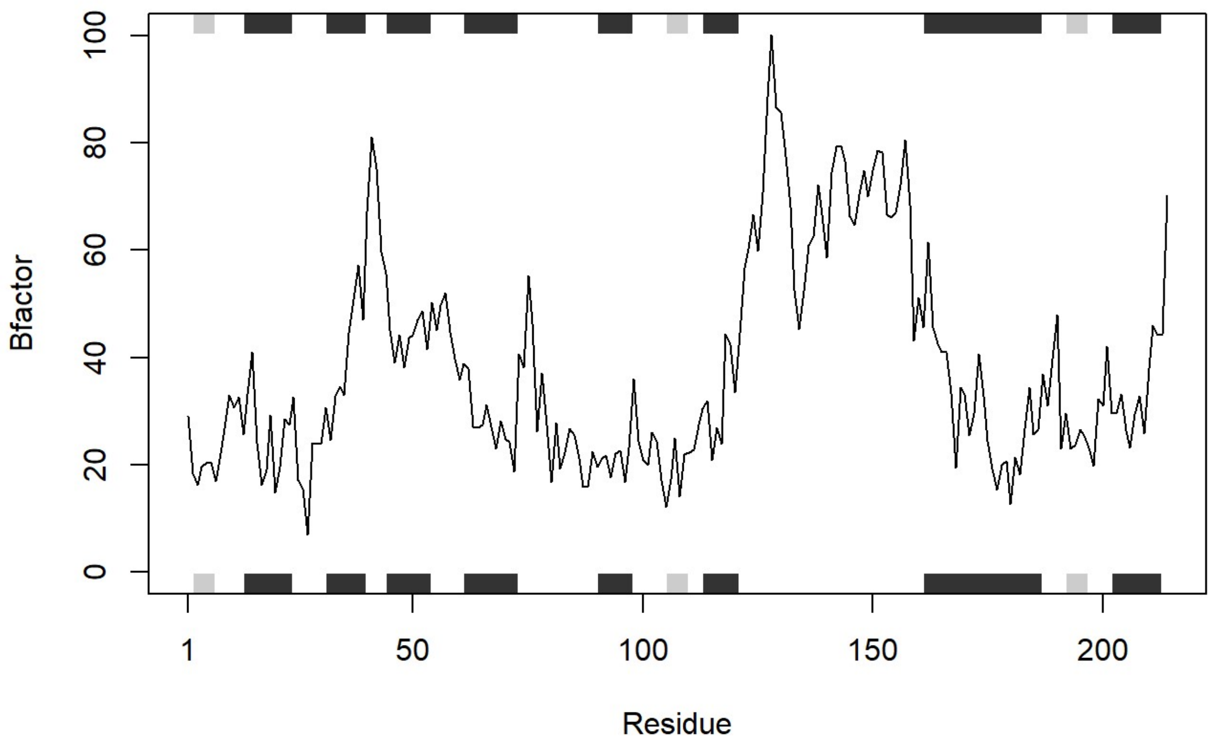
Q6: How would you generalize the original code above to work with any set of input protein structures?

```
# Function takes in a 4-letter protein identifier as it's only argument. Accesses pdb
database and returns data frame of all protein information stored in protein. protein
is then passed into trim.pdb which saves only the A chain and a character vector of at
om names into protein.chainA. This trimmed down vector then subsets out the b column o
f vector "atom" and saves it to protein.b. This is then plotted and returns a line plo
t that displays the Bfactor activity of a particular protein.

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

Testing the function

```
# Calling the optimized function
proteinActivityPlot("4AKE")
```

```
##   Note: Accessing on-line PDB file
## Warning in get.pdb(file, path = tempdir(), verbose = FALSE): C:
## \Users\CHRIST~1\AppData\Local\Temp\RtmpO0oOJc/4AKE.pdb exists. Skipping download
```

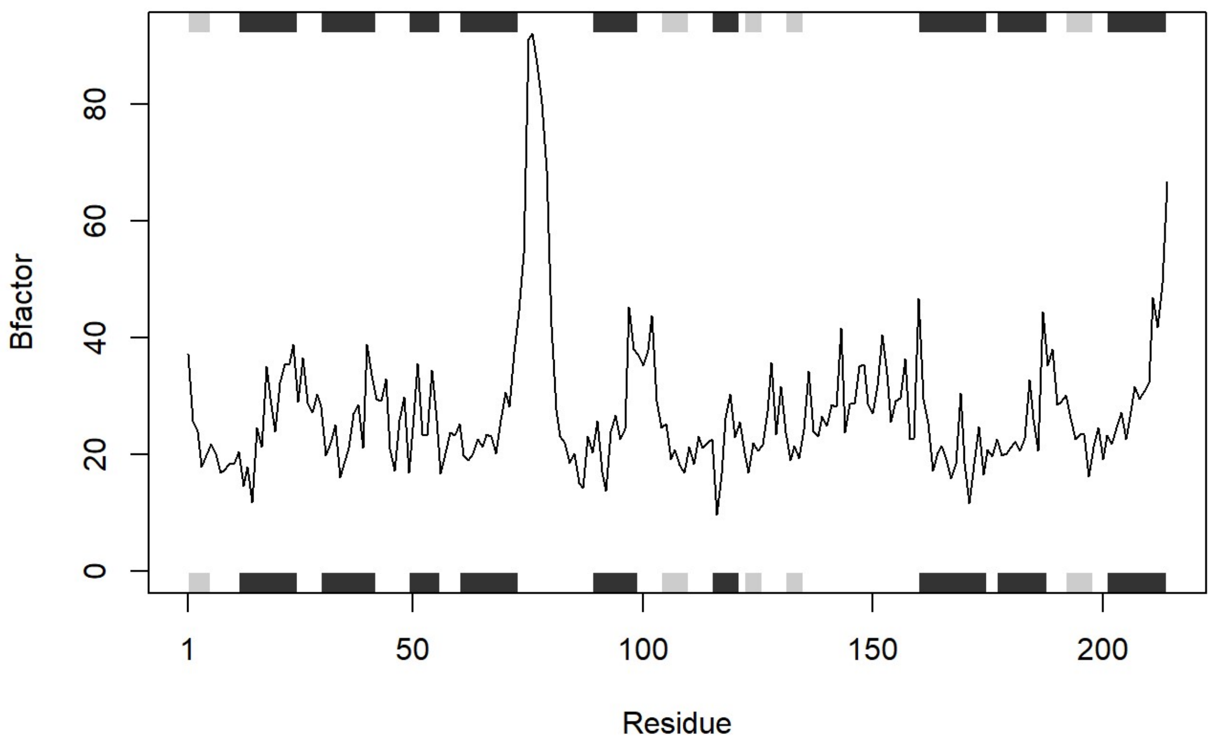


```
proteinActivityPlot("1AKE")
```

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

```
## Warning in get.pdb(file, path = tempdir(), verbose = FALSE): C:
## \Users\CHRIST~1\AppData\Local\Temp\RtmpO0oOJc/1AKE.pdb exists. Skipping download
```

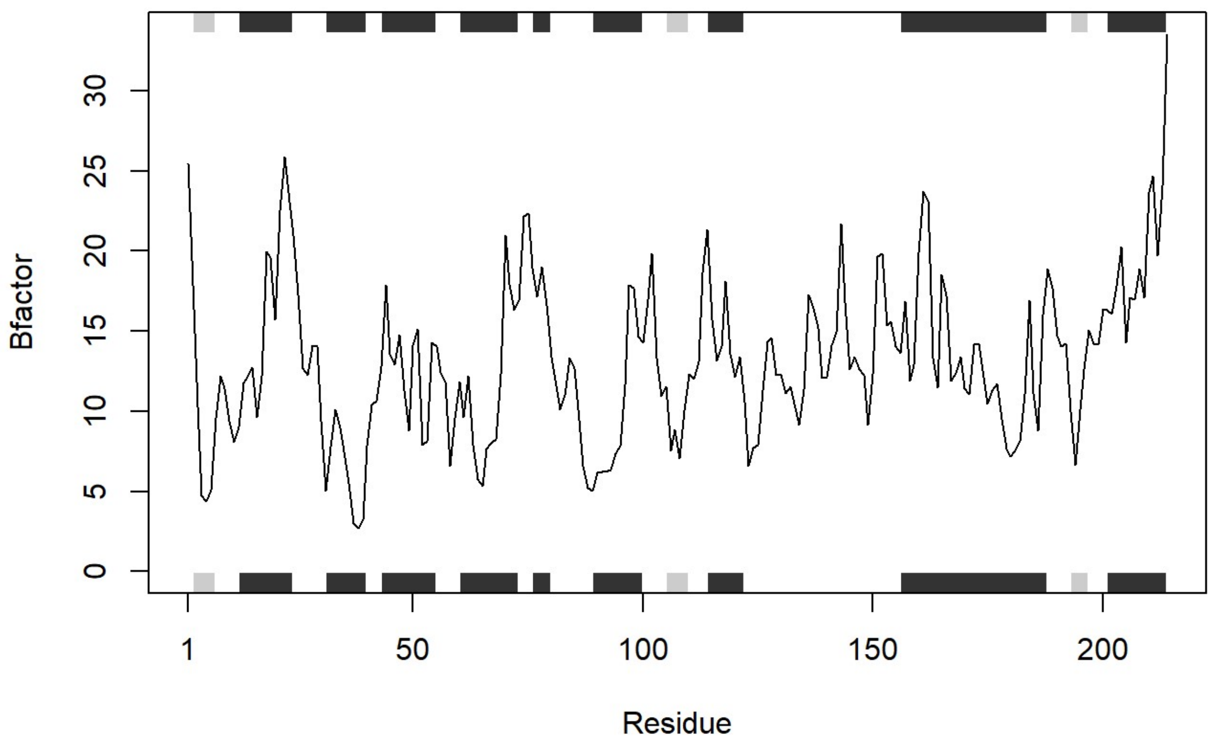
```
##    PDB has ALT records, taking A only, rm.alt=TRUE
```



```
proteinActivityPlot("1E4Y")
```

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

```
## Warning in get.pdb(file, path = tempdir(), verbose = FALSE): C:
## \Users\CHRIST~1\AppData\Local\Temp\RtmpO0oOJc/1E4Y.pdb exists. Skipping download
```



Bfactor
0
5
10
15
20
25
30
1
50
100
150
200
Residue