Improving analysis code by writing functions

Andrew Kapinos

10/23/2021

Can you improve this analysis code?

```r
# library(bio3d)
# s1 <- read.pdb("4AKE") # kinase with drug
# s2 <- read.pdb("1AKE") # kinase no drug
# s3 <- read.pdb("1E4Y") # kinase with drug

# 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")
```

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

We need to break down the original code into its various steps.

Step 1: Loads bio3d packages.
Step 2: Accesses online PDB file and assigns all data to an object.
Step 3: Trims PDB file to only include chain A alpha-carbons and assigns filtered data to an object.
Step 4: Describes B-factor data for chain A alpha-carbons.
Step 5: Plots B-factor data.

Starting at step 2, we can generalize the code given for accessing the PDB file and assigning its output to an object.

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

# Simplified version
library(bio3d)
x <- "4AKE"
pdb_data <- read.pdb(x)
```

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

Moving onto step 3, let's generalize the code used to trim the PDB file to include only chain A alpha-carbons.

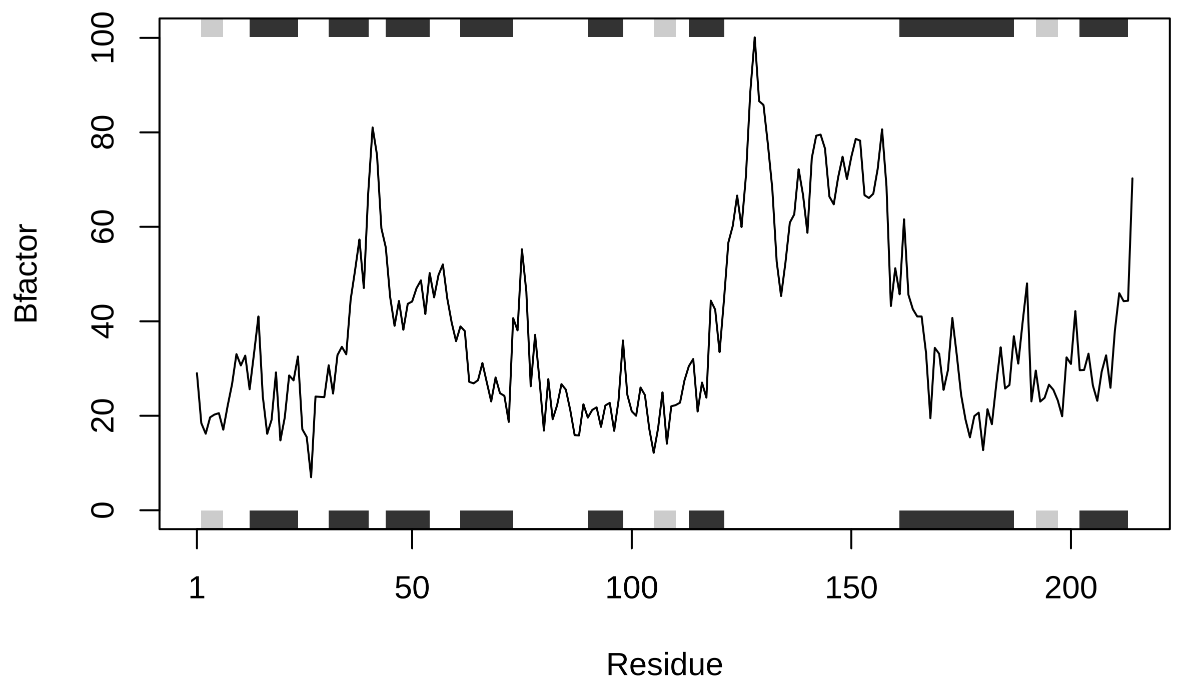
```
# Original code
# s1.chainA <- trim.pdb(s1, chain="A", elety="CA")

# Simplified version
chainA <- trim.pdb(pdb_data, chain="A", elety="CA")
```

Moving onto step 4, let's generalize the code used to describe *B-factor* for alpha-carbons in chain A.

```
# Original code
# s1.b <- s1.chainA$atom$b

# Simplified version
b <- chainA$atom$b
```

Finally, in step 5, we can generalize the code used to plot the B-factor data.

```
# Original code
# plotb3(s1.b, sse=s1.chainA, typ="l", ylab="Bfactor")

# Simplified version
plotb3(b, sse=chainA, typ="l", ylab="Bfactor")
```

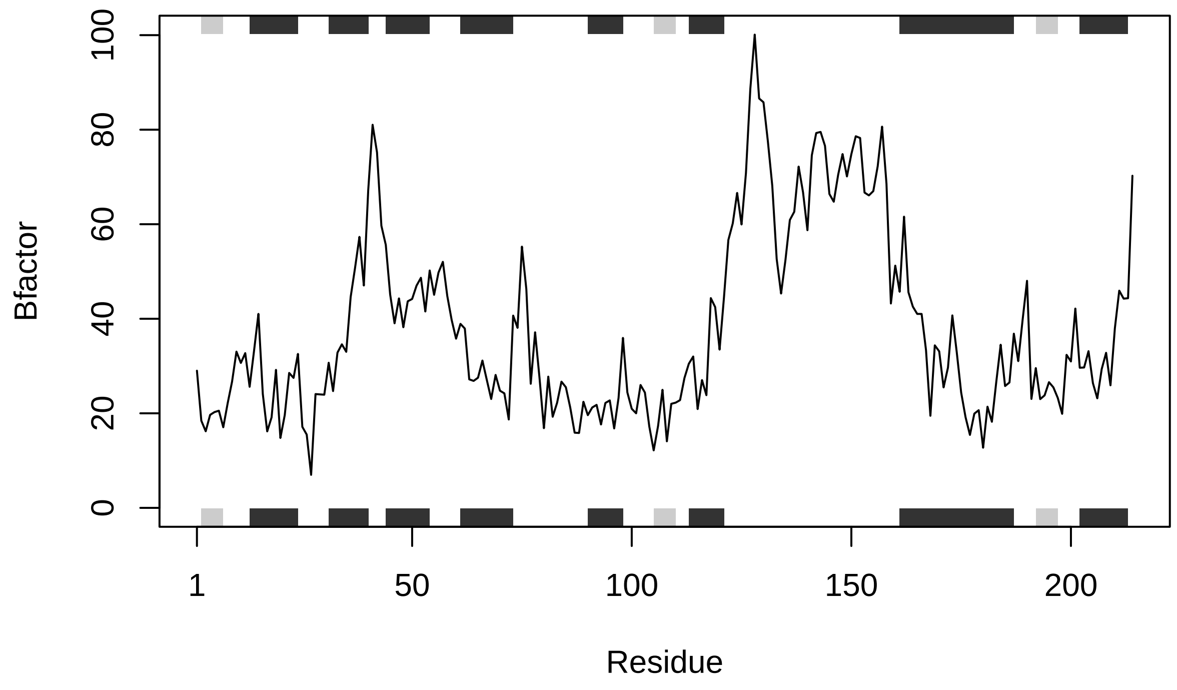


Let's put our generalized pieces all together!

```
x <- "4AKE"
library(bio3d)
pdb_data <- read.pdb(x)
```

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

## Warning in get.pdb(file, path = tempdir(), verbose = FALSE): /var/folders/08/
## v95p5lpj0c1dymxdpt1292pw0000gn/T//Rtmp0ovmVq/4AKE.pdb exists. Skipping download
```

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



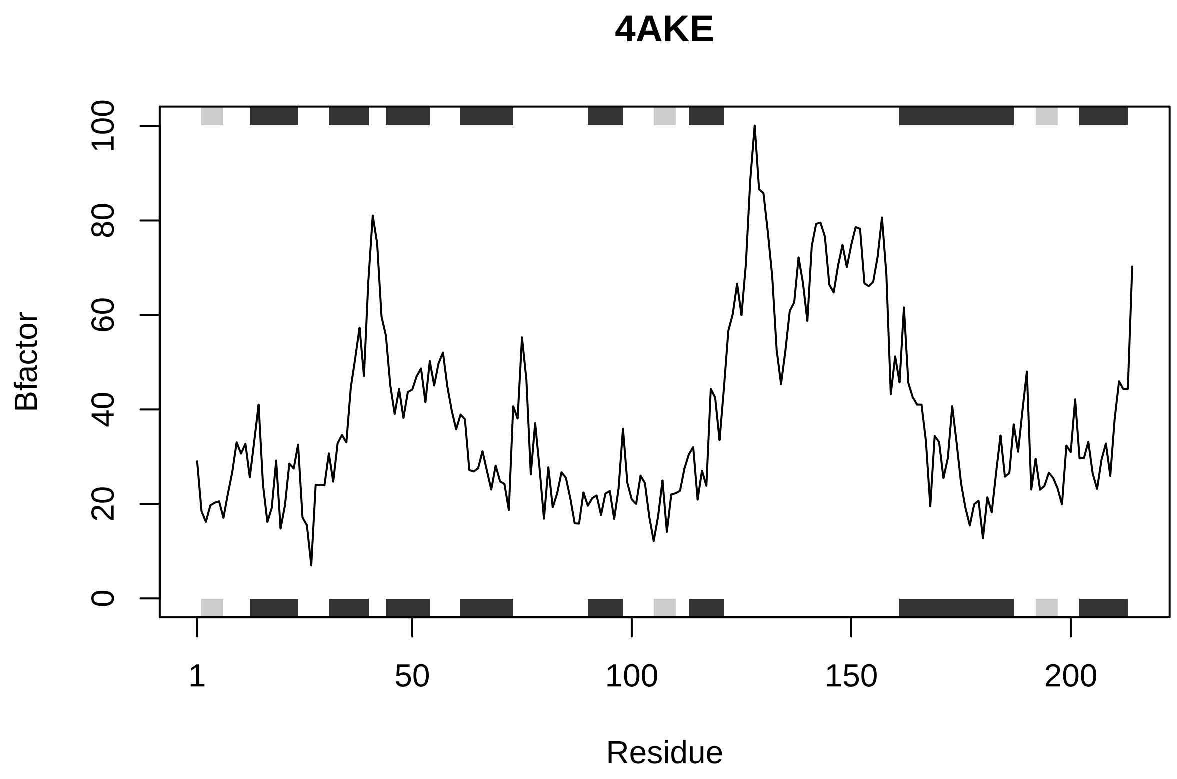
Now, let's write our code into a function. I've added an optional argument, **main=x** to the plotb3() function, which will add the PDB code to each output graph as a plot title. This will make it easier to keep track of our output plots if we are using apply() to run the function on a data frame containing PDB IDs.

```
prot_drug_interaction <- function(x) {
  #Load bio3d package if not already done
  library(bio3d)

  # Access PDB using an input PDB ID, "x", and store data to an object
  pdb_data <- read.pdb(x)
  # Trim the PDB info to only include the data for alpha-carbons in chain A
  chainA <- trim.pdb(pdb_data, chain="A", elety="CA")

  # Create a vector with B-factor data for alpha-carbons in chain A
  b <- chainA$atom$b

  # Plot the B-factor data for alpha-carbons in chain A
  plotb3(b, sse=chainA, typ="l", ylab="Bfactor",main=x)
}
```

And let's verify that it works for one of our sample inputs. To use the function, we only need to provide the function with the PDB ID code of the protein we are interested in.

```
prot_drug_interaction("4AKE")
```

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

## Warning in get.pdb(file, path = tempdir(), verbose = FALSE): /var/folders/08/
## v95p5lpj0c1dymxdpt1292pw0000gn/T//Rtmp0ovmVq/4AKE.pdb exists. Skipping download
```



As we can see, the output of the function is a line plot displaying B-factor data for alpha-carbons in chain A of the protein of interest.

Let's create a data frame that contains the PDB codes provided as example inputs above. Then, we can use *apply()* to run the function on each of the original input PDB codes, by checking each row in the data frame for the input, x (ie. the PDB code).

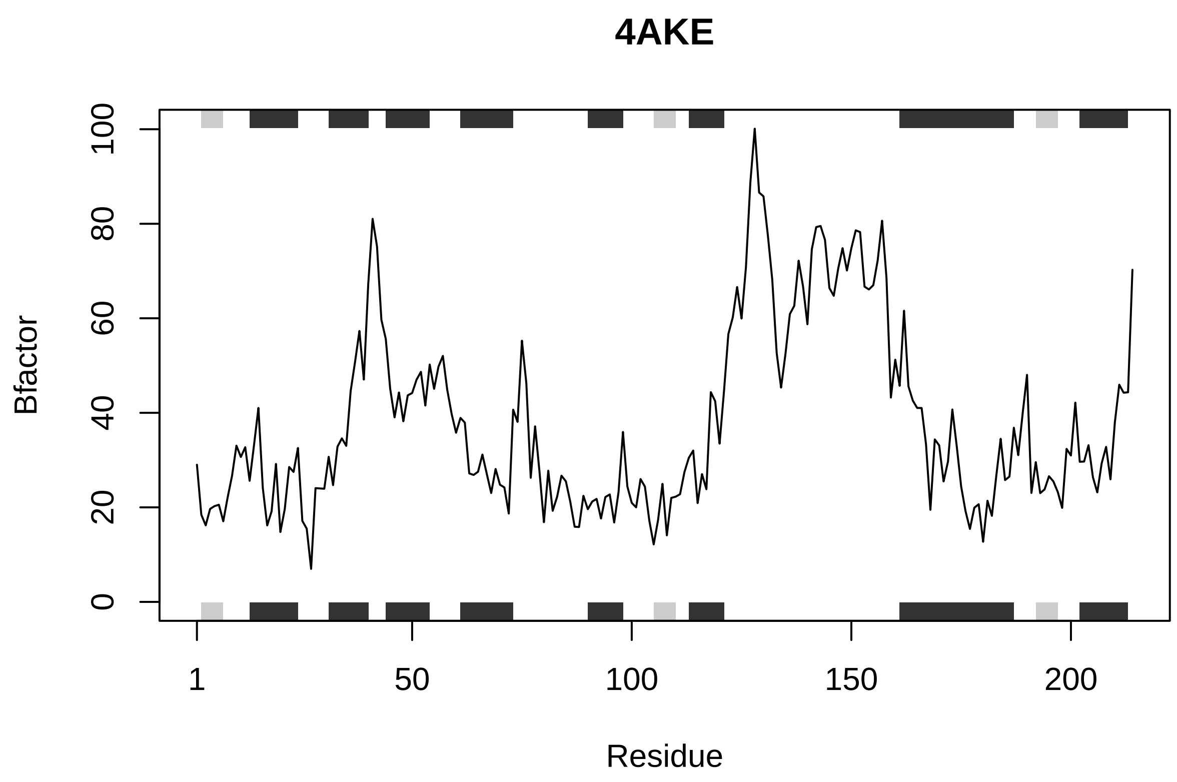
```r
PDB_codes <- data.frame(c("4AKE","1AKE","1E4Y"))
colnames(PDB_codes) <- "PDB_ID"
PDB_codes
```

```
##   PDB_ID
## 1   4AKE
## 2   1AKE
## 3   1E4Y
```

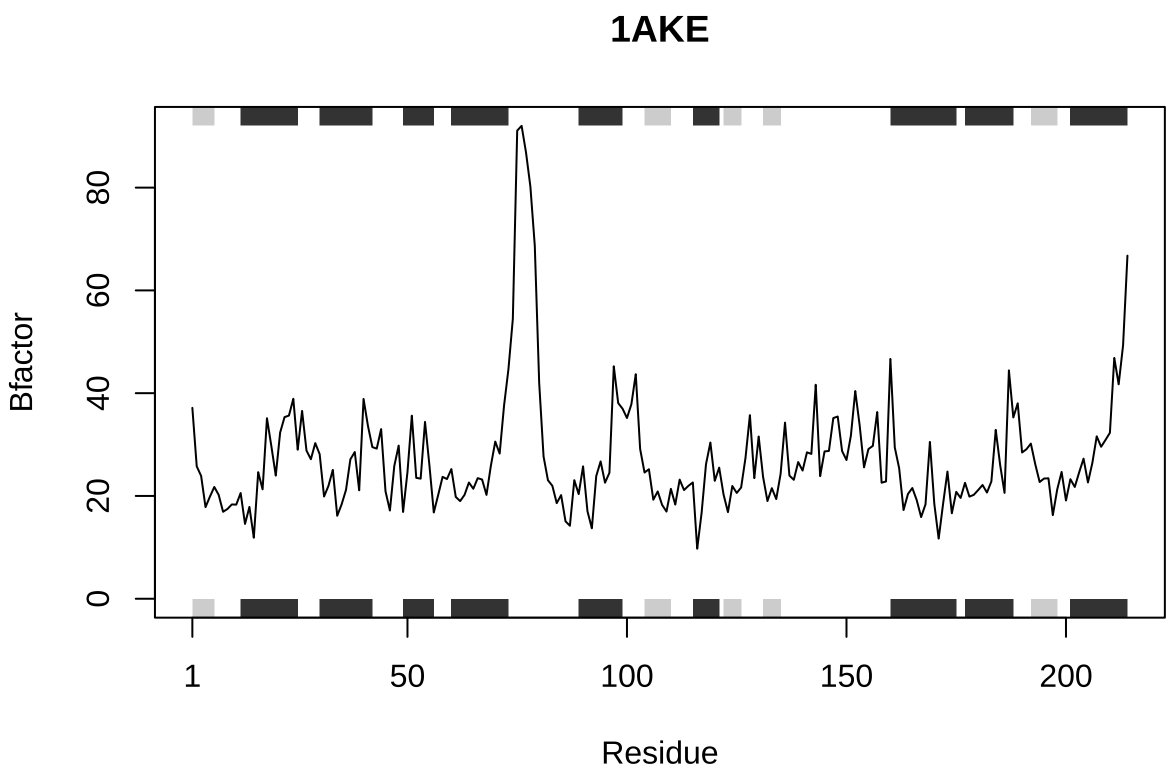
```r
apply(PDB_codes,1,prot_drug_interaction)
```

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

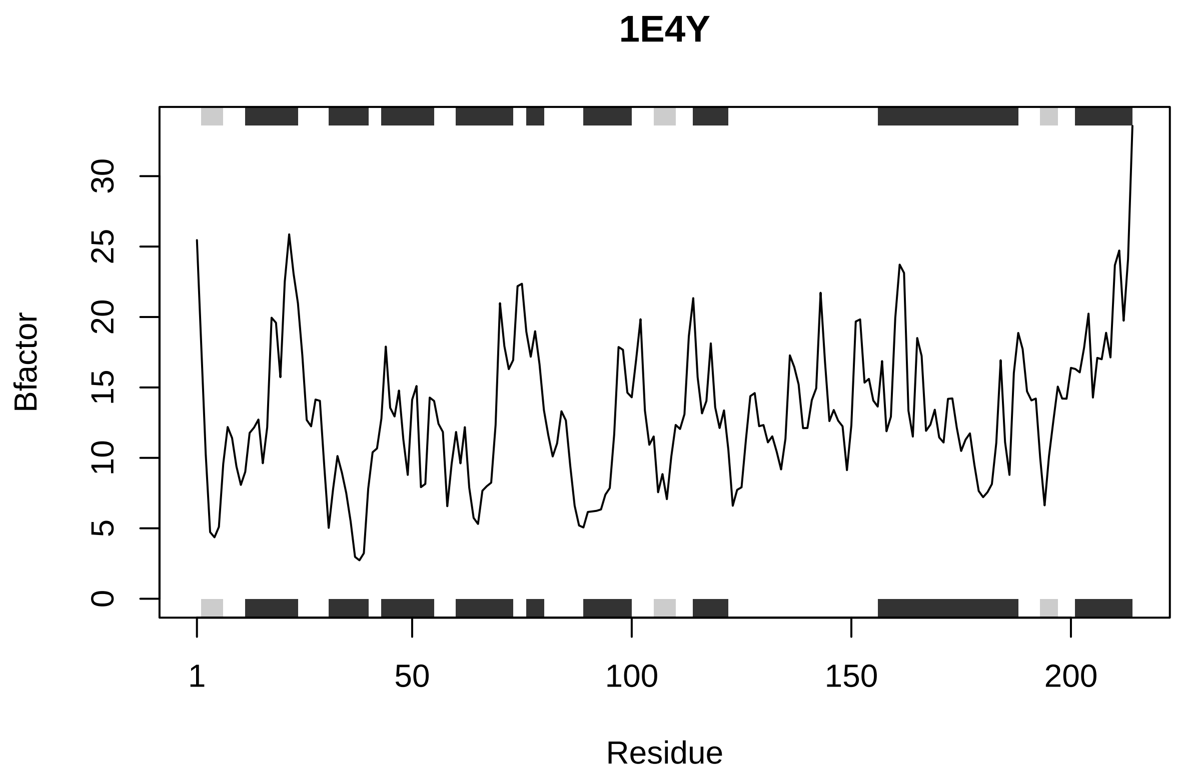
```
## Warning in get.pdb(file, path = tempdir(), verbose = FALSE): /var/folders/08/
## v95p5lpj0c1dymxdpt1292pw0000gn/T//Rtmp0ovmVq/4AKE.pdb exists. Skipping download
```



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



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



```
## NULL
```