Practice Assignment

The goal of this assignment is to provide a “bridge” between the first two weeks of lectures and assignment 1 for those either new to R or struggling with how to approach the assignment.

This guided example, will **not** provide an “solution” for programming assignment 1. However, it will guide you through some core concepts and give you some practical experience to hopefully make assignment 1 seems less daunting.

To begin, download this file and unzip it into your R working directory.
https://dl.dropboxusercontent.com/u/8036886/diet_data.zip

You can do this in R with the following code:

```
dataset_url <- "http://dl.dropboxusercontent.com/u/8036886/diet_data.zip"
download.file(dataset_url, "diet_data.zip")
unzip("diet_data.zip", exdir = "diet_data")
```

If you're not sure where your working directory is, you can find out with the getwd() command. Alternatively, you can view/change it through the Tools > Global Options menu in R Studio.

So assuming you've unzipped the file into your R directory, you should have a folder called diet_data. In that folder there are five files. Let's get a list of those files:

```
list.files("diet_data")
```

```
## [1] "Andy.csv"  "David.csv" "John.csv"  "Mike.csv"  "Steve.csv"
```

Okay, so we have 5 files. Let's take a look at one to see what's inside:

```
andy <- read.csv("diet_data/Andy.csv")
head(andy)
```

```
##   Patient.Name Age Weight Day
## 1         Andy  30    140   1
## 2         Andy  30    140   2
## 3         Andy  30    140   3
## 4         Andy  30    139   4
## 5         Andy  30    138   5
## 6         Andy  30    138   6
```

It appears that the file has 4 columns, Patient.Name, Age, Weight, and Day. Let's figure out how many rows there are by looking at the length of the 'Day' column:

```
length(andy$Day)
```

```
## [1] 30
```

30 rows. OK.

Alternatively, you could look at the dimensions of the data.frame:

```
dim(andy)
```

```
## [1] 30  4
```

This tells us that we 30 rows of data in 4 columns.

So we have 30 days of data. To save you time, all of the other files match this format and length. I've made up 30 days worth of weight data for 5 subjects of an imaginary diet study.

Let's play around with a couple of concepts. First, how would we see Andy's starting weight? We want to subset the data. Specifically, the first row of the 'Weight' column:

```
andy[1, "Weight"]
```

```
## [1] 140
```

We can do the same thing to find his final weight on Day 30:

```
andy[30, "Weight"]
```

```
## [1] 135
```

Alternatively, you could create a subset of the 'Weight' column where the data where 'Day' is equal to 30.

```
andy[andy$Day == 30, "Weight"]
```

```
## [1] 135
```

```
andy[andy[, "Day"] == 30, "Weight"]
```

```
## [1] 135
```

There are lots of ways to get from A to B when using R. However it's important to understand some of the various approaches to subsetting data.

Let's assign Andy's starting and ending weight to vectors:

```
andy_start <- andy[1, "Weight"]
andy_end <- andy[30, "Weight"]
```

We can find out how much weight he lost by subtracting the vectors:

```
andy_loss <- andy_start - andy_end
andy_loss
```

```
## [1] 5
```

Andy lost 5 pounds over the 30 days. Not bad. What if we want to look at other subjects or maybe even everybody at once?

Let's look back to the list.files() command. It returns the contents of a directory in alphabetical order. You can type '?list.files' at the R prompt to learn more about the function.

Let's take the output of list.files() and store it:

```
files <- list.files("diet_data")
files
```

```
## [1] "Andy.csv"  "David.csv" "John.csv"  "Mike.csv"  "Steve.csv"
```

Knowing that 'files' is now a list of the contents of 'diet_data' in alphabetical order, we can call a specific file by subsetting it:

```
files[1]
```

```
## [1] "Andy.csv"
```

```
files[2]
```

```
## [1] "David.csv"
```

```
files[3:5]
```

```
## [1] "John.csv"  "Mike.csv"  "Steve.csv"
```

Let's take a quick look at John.csv:

```
head(read.csv(files[3]))
```

```
## Warning: cannot open file 'John.csv': No such file or directory
```

```
## Error: cannot open the connection
```

Woah, what happened? Well, John.csv is sitting inside the diet_data folder. We just tried to run the equivalent of read.csv("John.csv") and R correctly told us that there isn't a file called John.csv in our working directory. To fix this, we need to append the directory to the beginning of the file name.

One approach would be to use paste() or sprintf(). However, if you go back to the help file for list.files(), you'll see that there is an argument called full.names that will append (technically prepend) the path to the file name for us.

```
files_full <- list.files("diet_data", full.names = TRUE)
files_full
```

```
## [1] "diet_data/Andy.csv"  "diet_data/David.csv" "diet_data/John.csv"
## [4] "diet_data/Mike.csv"  "diet_data/Steve.csv"
```

Pretty cool. Now let's try taking a look at John.csv again:

```
head(read.csv(files_full[3]))
```

```
##   Patient.Name Age Weight Day
## 1         John  22    175   1
## 2         John  22    175   2
## 3         John  22    175   3
## 4         John  22    175   4
## 5         John  22    175   5
## 6         John  22    175   6
```

Success! So what if we wanted to create one big data frame with everybody's data in it? We'd do that with rbind and a loop. Let's start with rbind:

```
andy_david <- rbind(andy, read.csv(files_full[2]))
```

This line of code took our existing data frame, Andy, and added the rows from David.csv to the end of it. We can check this with:

```
head(andy_david)
```

```
##   Patient.Name Age Weight Day
## 1         Andy  30    140   1
## 2         Andy  30    140   2
## 3         Andy  30    140   3
## 4         Andy  30    139   4
## 5         Andy  30    138   5
## 6         Andy  30    138   6
```

```
tail(andy_david)
```

```
##    Patient.Name Age Weight Day
## 55        David  35    203  25
## 56        David  35    203  26
## 57        David  35    202  27
## 58        David  35    202  28
## 59        David  35    202  29
## 60        David  35    201  30
```

One thing to note, rbind needs 2 arguments. The first is an existing data frame and the second is what you want to append to it. This means that there are occassions when you might want to create an empty data frame just so there's *something* to use as the existing data frame in the rbind argument.

Don't worry if you can't imagine when that would be useful because you'll see an example in just a little while.

Now, let's create a subset of the data frame that shows us just the 25th day for Andy and David.

```
day_25 <- subset(andy_david, andy_david$Day == 25)
day_25
```

```
##    Patient.Name Age Weight Day
## 25         Andy  30    135  25
## 55        David  35    203  25
```

Now you could manually go through and append everybody's data to the same data frame using rbind, but that's not a practical solution if you've got lots and lots of files. So let's try using a loop.

To understand what's happening in a loop, let's try something:

```
for (i in 1:5) {
    print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

As you can see, for each pass through the loop, i increases by 1 from 1 through 5. Let's apply that concept to our list of files.

```
for (i in 1:5) {
    dat <- rbind(dat, read.csv(files_full[i]))
}
```

```
## Error: object 'dat' not found
```

Whoops. Object 'dat' not found. This is because you can't rbind something into a file that doesn't exist yet. So let's create an empty data frame called 'dat' before running the loop.

```
dat <- data.frame()
for (i in 1:5) {
    dat <- rbind(dat, read.csv(files_full[i]))
}
```

Cool. We now have a data frame called 'dat' with all of our data in it. So what if we wanted to know the median weight for all the data? Let's use the median() function.

```
median(dat$Weight)
```

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

NA? Why did that happen? Type 'dat' into the console and you'll see a print out of all 150 obversations. Scroll back up to row 77, and you'll see that we have some missing data from John, which is recorded as NA by R.

We need to get rid of those NA's for the purposes of calculating the median. There are several approaches. We could subset the data using complete.cases. But if you look at '?median', you'll see there is an argument called 'na.rm' that will strip the NA values out for us.

```
median(dat$Weight, na.rm = TRUE)
```

```
## [1] 190
```

So 190 is the median weight. We can find the median weight of day 30 by taking the median of a subset of the data where Day=30.

```
dat_30 <- dat[dat[, "Day"] == 30, ]
dat_30
```

```
##     Patient.Name Age Weight Day
## 30          Andy  30    135  30
## 60         David  35    201  30
## 90          John  22    177  30
## 120         Mike  40    192  30
## 150        Steve  55    214  30
```

```
median(dat_30$weight)
```

```
## [1] 192
```

We've done a lot of manual data manipulation so far. Let's build a function that will return the median weight of a given day.

Let's start out by defining what the arguments of the function should be. These are the parameters that the user will define. The first parameter the user will need to define is the directory that is holding the data. The second parameter they need to define is the day for which they want to calculate the median.

So our function is going to start out something like this:

weightmedian <- function(directory, day) { # content of the function }

So what goes in the content? Let's think through it logically. We need a data frame with all of the data from the CSV's. We'll then subset that data frame using the argument 'day' and take the median of that subset.

In order to get all of the data into a single data frame, we can use the method we worked through earlier using list.files and rbind.

Essentially, these are all things that we've done in this example. Now we just need to combine them into a single function.

So what does the function look like?

```
weightmedian <- function(directory, day) {
    files_list <- dir(directory, full.names = TRUE)  #creates a list of files
    dat <- data.frame()  #creates an empty data frame
    for (i in 1:5) {
        # loops through the files, rbinding them together
        dat <- rbind(dat, read.csv(files_list[i]))
    }
    dat_subset <- dat[dat[, "Day"] == day, ]  #subsets the rows that match the 'day' argument
    median(dat_subset$weight, na.rm = TRUE)  #identifies the median of the subset
    # while stripping out the NAs
}
```

You can test this function by running it a few different times:

```
weightmedian("diet_data", 20)
```

```
## [1] 197.5
```

```
weightmedian("diet_data", 4)
```

```
## [1] 188
```

```
weightmedian("diet_data", 17)
```

```
## [1] 198
```

Hopefully, this has given you some practice applying the basic concepts from weeks 1 and 2. If you can work your way through this example, you should have all of the tools needed to complete part 1 of assignment 1. Parts 2 and 3 are really just expanding on the same basic concepts, potentially adding in some ideas like cbinds and if-else.

I'm going to start a forum thread for this “practice assignment” and will do my best to answer questions.