# Diet Data Analysis in R: Exploring, Subsetting, and Combining Multiple CSV Files

Hassan Reza Rezaye

2025-08-09

This document is part of the R Programming course from the Johns Hopkins University Data Science Specialization on Coursera. Through this assignment, I gained a solid grasp of several useful R programming concepts. It reflects my understanding and may serve as a helpful reference for other aspiring learners of R.

The documentation starts with the download of a zipped file named `diet_data.zip` containing five datasets; each provides statistics about five patients and focuses on weight in a 30 days time period. I then move on to explore the file and each datasets applying several ways of subsettings to access specific and a desired part of the data. Accordingly, I showed how to generate a customized function that yields an expacted result like `median` of a variable in a dataset. I also elaborated on ways of reading csv files individually and multiple file at once. Each way involves functions like `rbind` and `loop`. At last, I end the document with the most effective and more advanced-like function called `do.call()` which enables a user to read dozens of files at once with ease.

I start the session with downloading a `diet_data.zip` and unzipping it in a folder called `data_diet`.

```r
# Downloading the dataset

dataset_url <- "http://s3.amazonaws.com/practice_assignment/diet_data.zip"
download.file(dataset_url, "diet_data.zip")
unzip("diet_data.zip", exdir = "data_diet")

# `exdir` extract the content of the zip file into a new directory/folder named data_diet.
```

## Step 2: Explore Directory and Files:

In the beginning, we can ensure ourselves about out working place or directory. Then check out if there the unzipped folder exists.

```r
getwd()
```

```
## [1] "C:/Users/test/OneDrive/Desktop"
```

```r
list.files("data_diet")      # there appears to be five files.
```

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

```r
file.exists("data_diet")    # The response is "TRUE" which mean it lives there.
```

```
## [1] TRUE
```

```r
list.files("data_diet")
```

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

```r
file.exists("data_diet")
```

```
## [1] TRUE
```

## Step 3: Read a CSV File and Explore Its Structure

lets take a look at one of the file `Steve.csv` and find out about its stacture and datatypes. It indicates that there are four variables or columns in the csv that includes: Patient.name, Age, Weight and Day.

```r
Steve <- read.csv("data_diet/steve.csv")
head(Steve)
```

```
##   Patient.Name Age Weight Day
## 1        Steve  55    225   1
## 2        Steve  55    225   2
## 3        Steve  55    225   3
## 4        Steve  55    224   4
## 5        Steve  55    224   5
## 6        Steve  55    224   6
```

```r
length(Steve$Day)    # The result show 30 rows.
```

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

```r
dim(Steve)           # The output for dim is 30 rows and 4 columns
```

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

```r
# The commands below gives us further feels about our dataset.

summary(Steve)       #it provide some basic aggregations
```

```
##  Patient.Name            Age         Weight           Day
##  Length:30          Min.   :55   Min.   :214.0   Min.   : 1.00
##  Class :character   1st Qu.:55   1st Qu.:217.0   1st Qu.: 8.25
##  Mode  :character   Median :55   Median :220.5   Median :15.50
##                     Mean   :55   Mean   :220.0   Mean   :15.50
##                     3rd Qu.:55   3rd Qu.:223.0   3rd Qu.:22.75
##                     Max.   :55   Max.   :225.0   Max.   :30.00
```

```r
str(Steve)           #it illustrate the structure of the dataset and
```

```
## 'data.frame':    30 obs. of  4 variables:
##  $ Patient.Name: chr  "Steve" "Steve" "Steve" "Steve" ...
##  $ Age         : int  55 55 55 55 55 55 55 55 55 55 ...
##  $ Weight      : int  225 225 225 224 224 224 223 223 223 223 ...
##  $ Day         : int  1 2 3 4 5 6 7 8 9 10 ...
```

```r
                     #the type of variables we are working with.

names(Steve)         #it lists the column/variable names of the dataset.
```

```
## [1] "Patient.Name" "Age"          "Weight"       "Day"
```

This data set display quantitative information about patient (Steve) weight through out 30 days and the other five data set for different patient is similar.

## Step 4: Accessing Specific Data:

lets play around with couple of concpets. Though few ways, we filter or subset out data to access a specific data. we check on dataset's first row and last last row separately to check on Steves weight in the beginning and end period.

```r
Steve[1, "Weight"]   # Steve's weight was 225 in the first row (in the begining)
```

```
## [1] 225
```

```r
                     # and 214 in the last row (in the end).

Steve[30, "Weight"]
```

```
## [1] 214
```

Another approach is to create a subset of weight where the "Day" is equal to 30. We use `which()` function which involves condition and outputs exact value we ask for.

```r
# Used `which()` to create a subset of weight where the "Day" is equal to 30.
Steve[which(Steve$Day == 30), "Weight"]
```

```
## [1] 214
```

```r
#Or
Steve[which(Steve[, "Day"] == 30), "Weight"]
```

```
## [1] 214
```

```r
#or
subset(Steve$Weight, Steve$Day == 30)      #In each way the weight is 214 in the Day 30.
```

```
## [1] 214
```

Now, we assign Steve's starting and ending weight to a vector of first day and last day weight to find the difference of weight (weigth loss).

```r
Steve_start <- Steve[1, "Weight"]
Steve_end <- Steve[30, "Weight"]
steve_loss <- Steve_start - Steve_end
steve_loss # Steve's gained a weight loss of 11 pound in 30 days.
```

```
## [1] 11
```

## Step 5: List and Read Multiple Files

Now, What if we want to look at other datasets or individual's weight at once. We can achieve this result in different way which we are going to explore now.

In the begining, lets see what the function `List.files()` do. It mainly lists all the contents existing in the working directory or the project folder we are working on.

We are now going to assign a character vector of files/datasets from the "data_diet" to the "files_diet" in alphabetic order. It mean we just copy the names not exactly the datasets. We want try to read each datasets one by one or couple of them at once by subsetting.

```r
files_diet <- list.files("data_diet")
files_diet[1]        # This subsets or outputs the first element in the `files_diet`
```

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

```r
files_diet[2:5]
```

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

Next, lets read one of the datasets from this vector list of content.

```r
#head(read.csv(files_diet[3]))         # Results error
```

This operation resulted in an error saying that the 3rd file (John.csv) dataset does not exist in the "files_diet" to be read. it means we just copied the names from "data_diet" not the dataset paths to the this vector to make it prepared for reading.

To make this work, we need to append or specify their full path or directory to "files_diet", a list of csv files which will be placed in alphabatic order.

```r
# the argument "full.names = TRUE" allow us to transfer the full path directory
# of the datasets to this vecotor list - `files_diet `.
files_diet1 <- list.files("data_diet", full.names = TRUE)
files_diet1
```

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

```
head(read.csv(files_diet1[3]))  # Success! by looking the previous dataset (john.csv),
```

```
##   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
```

```
                                  # we could read.
```

## Step 7: Combine Data with `rbind()`

Here we go through next step, In this step we are going to create a big data frame by combining each dataset or each patients data file using `rbind()` and loop.

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 occasions when you might want to create an empty data frame just to combine the existing data frame in the rbind argument.

```
# I read "And.csv" the first element of the `files_diet1` and then
# combined to "Steve.csv" row wise.
Steve_Andy <- rbind(Steve, read.csv(files_diet1[1]))
head(Steve_Andy)
```

```
##   Patient.Name Age Weight Day
## 1        Steve  55    225   1
## 2        Steve  55    225   2
## 3        Steve  55    225   3
## 4        Steve  55    224   4
## 5        Steve  55    224   5
## 6        Steve  55    224   6
```

```
tail(Steve_Andy)
```

```
##    Patient.Name Age Weight Day
## 55         Andy  30    135  25
## 56         Andy  30    135  26
## 57         Andy  30    135  27
## 58         Andy  30    135  28
## 59         Andy  30    135  29
## 60         Andy  30    135  30
```

```
# Here I created a subset of dataframe that shows us 20th day for both Steve and Andy.
day_20 <- Steve_Andy[which(Steve_Andy$Day == 20), ]
day_20
```

```
##    Patient.Name Age Weight Day
## 20        Steve  55    219  20
## 50         Andy  30    137  20
```

## Step 8: Using Loops for Multiple Files

What if we create a bigger data frame and add each patient's data into it. Tt is possible to achieve it just by `rbind()` but too time consuming since we have to append each dataset manually. Therefor, we can use `loop` to combine lots of files into a single big data frame at once.

The syntax of loop looks like `for (i in 1:5) {print(i)}`. This syntax says; for each index (i), loops over the range (1:5) and apply the concept in {} to the list of element in `data_diet1`.

```
for (i in 1:5) { print(i) }   # This is loop function in general.
```

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

```
#This empty data frame `dat0` has to be created first otherwise we will face error.
dat0 <- data.frame()
for (i in 1:5) {
  dat0 <- rbind(dat0, read.csv(files_diet1[i]))
# It works now. We have each of the five datasets in a single dataframe `dat0`.
}
str(dat0)
```

```
## 'data.frame':    150 obs. of  4 variables:
##  $ Patient.Name: chr  "Andy" "Andy" "Andy" "Andy" ...
##  $ Age         : int  30 30 30 30 30 30 30 30 30 30 ...
##  $ Weight      : int  140 140 140 139 138 138 138 138 138 138 ...
##  $ Day         : int  1 2 3 4 5 6 7 8 9 10 ...
```

Question - what if we created `dat0` inside loop function? In this test, I named the empty data frame `dat1`.

```
for (i in 1:5) {
  dat1 <- data.frame()
  dat1 <- rbind(dat1, read.csv(files_diet1[i]))
}
str(dat1)
```

```
## 'data.frame':    30 obs. of  4 variables:
##  $ Patient.Name: chr  "Steve" "Steve" "Steve" "Steve" ...
##  $ Age         : int  55 55 55 55 55 55 55 55 55 55 ...
##  $ Weight      : int  225 225 225 224 224 224 223 223 223 223 ...
##  $ Day         : int  1 2 3 4 5 6 7 8 9 10 ...
```

```
head(dat1)
```

```
##   Patient.Name Age Weight Day
## 1        Steve  55    225   1
## 2        Steve  55    225   2
## 3        Steve  55    225   3
## 4        Steve  55    224   4
## 5        Steve  55    224   5
## 6        Steve  55    224   6
```

We found out that the loop resulted in the last dataset `Steve.csv` from “files_diet”. It is because every time it loops, for each index in the given range, it rewrite the “dat1” data frame with each pass of the loop.

## Step 9: Analyze Weight Data

Lets work on figuring out median weight for all the `dat1` using median() function.

```
median(dat0$Weight)  # we got the result NA. It is because the data set
```

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

```
                     # contain missing values. Hence, we need to sort it out
                     # in the median() by adding an argument `na.rm = TRUE`.

median(dat0$Weight, na.rm = TRUE)# It works now. the median weight results in 220.5 pound
```

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

```
                                 # for the entire dataset in the `dat0` data frame.

day_30 <- dat0[which(dat1$Day == 30), ] # Applied subset/filter to the data frame,
                                        # we can find out median weight for day 30.
day_30
```

```
##    Patient.Name Age Weight Day
## 30         Andy  30    135  30
```

```
median(day_30$Weight)
```

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

## Step 10: Create a Custom Function

Here we hace reached to the interesting part, creating a function that returns the median weight of any given day. To formulate a function, we need to specify the what argument a function can use which is defined by the users while applying the application. The argument in the function orderly include: directory/folder and day.

The prospective Function looks like: `weightmedian <- funtion(directory, day) { #content of the function }`

How do we define the content of the function?

lets think this way. We need an empty data frame `dat` in which we will transfer all of the CSV files by path applying `loop`. We’ll then subset that data frame using the argument `day` and take the median of that subset. To achieve this we apply what went through earlier wich is `file.list()` and `rbind()`. These are the tool we need to apply them collectively now.

```
weightmedian <- function(directory, day) {
  files_diet <- list.files(directory, full.names = TRUE) # a vector list of content from
  dat <- data.frame()                                    # `dat` is an empty data frame to
                                                         # store all the combined datasets
        for (i in 1:5) {
                  dat <- rbind(dat, read.csv(files_diet[i]))
                  # loop is used to combine the dataset with `rbind` function into `dat`.
        }
        dat_subset <- dat[which(dat[, "Day"] == day ), ] # subseting the rows based on the day.
          median(dat_subset[, "Weight"], na.rm = TRUE)  # the median weight on
                                                        # the given day while avoiding NAs.

}

weightmedian("data_diet", 20)
```

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

```
weightmedian("data_diet", 5)
```

```
## [1] 189
```

```
weightmedian("data_diet", 25)
```

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

## Step 11: A Better Approach Using Lists and 'do.call()

Up to this point, I primarily worked on how to read multiple file individually and collectively. Using `loop` beside `rbind` found out to be another way to read many file at once by building vector and a data frame inside a loop. Additionally, We also used a loop to generate a customized function to get a desired result like median from such big data frame.

Worth to not that they have their limitation which is continuously re-copying files in a data fram inside a loop as the files keep increasing. It is time consuming and frustrating.

Here, I am stepping on the next approach to find a better way that makes our work much easier to combine dozen and dozen of files at once and read it quickly.

The approach is to create an output object of an appropriate size and then fill it up. So the first thing we do is to create an empty list that's the length of our expected output. In this case, our input object is going to be `files_diet1` and our empty list is going to be `empty_list`.

```
# Created an empty list to store objects of a length equal to the length of `files_diet1`.
empty_list <- vector(mode = "list", length = length(files_diet1))

#Now, lets read through each csv files, orderly, and drop them into `empty_list`. We use `loop` to do i
for (i in seq_along(files_diet1)) {
  empty_list[[i]] <- read.csv(files_diet1[[i]])
}
str(empty_list)
```

```
## List of 5
##  $ :'data.frame':    30 obs. of  4 variables:
##   ..$ Patient.Name: chr [1:30] "Andy" "Andy" "Andy" "Andy" ...
##   ..$ Age         : int [1:30] 30 30 30 30 30 30 30 30 30 30 ...
##   ..$ Weight      : int [1:30] 140 140 140 139 138 138 138 138 138 138 ...
##   ..$ Day         : int [1:30] 1 2 3 4 5 6 7 8 9 10 ...
##  $ :'data.frame':    30 obs. of  4 variables:
##   ..$ Patient.Name: chr [1:30] "David" "David" "David" "David" ...
##   ..$ Age         : int [1:30] 35 35 35 35 35 35 35 35 35 35 ...
##   ..$ Weight      : int [1:30] 210 209 209 209 209 209 209 208 208 208 ...
##   ..$ Day         : int [1:30] 1 2 3 4 5 6 7 8 9 10 ...
##  $ :'data.frame':    30 obs. of  4 variables:
##   ..$ Patient.Name: chr [1:30] "John" "John" "John" "John" ...
##   ..$ Age         : int [1:30] 22 22 22 22 22 22 22 22 22 22 ...
##   ..$ Weight      : int [1:30] 175 175 175 175 175 175 175 175 175 175 ...
##   ..$ Day         : int [1:30] 1 2 3 4 5 6 7 8 9 10 ...
##  $ :'data.frame':    30 obs. of  4 variables:
##   ..$ Patient.Name: chr [1:30] "Mike" "Mike" "Mike" "Mike" ...
##   ..$ Age         : int [1:30] 40 40 40 40 40 40 40 40 40 40 ...
##   ..$ Weight      : int [1:30] 188 188 188 188 189 189 189 189 189 189 ...
##   ..$ Day         : int [1:30] 1 2 3 4 5 6 7 8 9 10 ...
##  $ :'data.frame':    30 obs. of  4 variables:
##   ..$ Patient.Name: chr [1:30] "Steve" "Steve" "Steve" "Steve" ...
##   ..$ Age         : int [1:30] 55 55 55 55 55 55 55 55 55 55 ...
##   ..$ Weight      : int [1:30] 225 225 225 224 224 224 223 223 223 223 ...
##   ..$ Day         : int [1:30] 1 2 3 4 5 6 7 8 9 10 ...
```

```r
summary(empty_list)
```

```
##      Length Class      Mode
## [1,] 4      data.frame list
## [2,] 4      data.frame list
## [3,] 4      data.frame list
## [4,] 4      data.frame list
## [5,] 4      data.frame list
```

We just applied `loop` to read each of the csv files and place them inside our `empty_list`. Now we have a list of 5 elements called `empty_list`, where each element of the list is a data frame containing one of the csv files. What we just did is functionally identical to using `lapply`.

```r
str(empty_list[[1]])
```

```
## 'data.frame':    30 obs. of  4 variables:
##  $ Patient.Name: chr  "Andy" "Andy" "Andy" "Andy" ...
##  $ Age         : int  30 30 30 30 30 30 30 30 30 30 ...
##  $ Weight      : int  140 140 140 139 138 138 138 138 138 138 ...
##  $ Day         : int  1 2 3 4 5 6 7 8 9 10 ...
```

```r
str(empty_list[1])
```

```
## List of 1
##  $ :'data.frame':    30 obs. of  4 variables:
```

```
##   ..$ Patient.Name: chr [1:30] "Andy" "Andy" "Andy" "Andy" ...
##   ..$ Age         : int [1:30] 30 30 30 30 30 30 30 30 30 30 ...
##   ..$ Weight      : int [1:30] 140 140 140 139 138 138 138 138 138 138 ...
##   ..$ Day         : int [1:30] 1 2 3 4 5 6 7 8 9 10 ...
```

```
head(empty_list[[1]][, "Day"])
```

```
## [1] 1 2 3 4 5 6
```

## we can combine each in a single dataframe.

Now, we still need to go from a list to a single data frame, although you *can* manipulate the data within this structure: We can use a function called `do.call()` to combine `emplya_list` elements into a single data frame. `do.call` lets you specify a function and then passes a list as if each element of the list were an argument to the function.

The syntax is `do.call(function_you_want_to_use, list_of_arguments)`. In our case, we want to `rbind()` our list of data frames which is in `empty_list`.

```
allfiles <- do.call(rbind, empty_list)
str(allfiles)
```

```
## 'data.frame':    150 obs. of  4 variables:
##  $ Patient.Name: chr  "Andy" "Andy" "Andy" "Andy" ...
##  $ Age         : int  30 30 30 30 30 30 30 30 30 30 ...
##  $ Weight      : int  140 140 140 139 138 138 138 138 138 138 ...
##  $ Day         : int  1 2 3 4 5 6 7 8 9 10 ...
```

This approach make reading magnitudes of csv file very comfortable by easily combine sperate data frames as a single big data frame with without frequent re-copying.

file.path()