R Notebook #2: Dive into Data Frames

Overview of data frames

A data frame is a two-dimensional table-like structure that can hold **columns of different types** (e.g., numeric, character, and logical). Each column in a data frame can be thought of as a vector, and the length of each vector (number of rows) must be the same across all columns.

```r
# Here, first I am loading an R "package" called "datasets." R packages are the building blocks on top
library(datasets)

# We can look at the structure mtcars with str() (Note this function works for other types of objects a
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

Let's look at the first 6 rows of the data frame mtcars, by using the function `head()`

```r
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

You can see that the car types do not have a column name. This is because the car names are not actually a column in the data frame. Instead, the rows also have names, which are the car types.

```r
# here we can check the column names to see the variables in the data frame
print("colnames mtcars:")
```

```
## [1] "colnames mtcars:"
```

```r
colnames(mtcars)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

```r
# here we can see the row names
print("rownames mtcars:")
```

```
## [1] "rownames mtcars:"
```

```r
rownames(mtcars)
```

```
##  [1] "Mazda RX4"           "Mazda RX4 Wag"       "Datsun 710"
##  [4] "Hornet 4 Drive"      "Hornet Sportabout"   "Valiant"
##  [7] "Duster 360"          "Merc 240D"           "Merc 230"
## [10] "Merc 280"            "Merc 280C"           "Merc 450SE"
## [13] "Merc 450SL"          "Merc 450SLC"         "Cadillac Fleetwood"
## [16] "Lincoln Continental" "Chrysler Imperial"   "Fiat 128"
## [19] "Honda Civic"         "Toyota Corolla"      "Toyota Corona"
## [22] "Dodge Challenger"    "AMC Javelin"         "Camaro Z28"
## [25] "Pontiac Firebird"    "Fiat X1-9"           "Porsche 914-2"
## [28] "Lotus Europa"        "Ford Pantera L"      "Ferrari Dino"
## [31] "Maserati Bora"       "Volvo 142E"
```

Navigating R data frames with base R

You can access a data frame's elements in several ways:

(1) **Like with matrices, we can use `[]` to designate a column.** But it is not a good coding practice to use numbers within the brackets. If our data frame gets rearranged, our code would be wrong. But we can use column names as follows:

```r
mpg <- mtcars["mpg"]  # here mpg is a data.frame with just one column
mpg <- mtcars[,"mpg"] # here mpg is a numeric vector
```

We can also use brackets to get particular rows. Note here, we need to put the row name before a comma. And it returns a data.frame.

```r
ferrari.dino <- mtcars["Ferrari Dino",] # a data.frame with just one row
```

Here's a little expansion in which I get all the Mercedes.

```r
# First I will create a vector that includes all Mazdas
Mazdas <- c("Mazda RX4","Mazda RX4 Wag")

# Then I will subset the data frame to the Mazdas
mtcars.Mazdas <- mtcars[Mazdas,]

print(mtcars.Mazdas)
```

```
##                 mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4        21   6  160 110  3.9 2.620 16.46  0  1    4    4
## Mazda RX4 Wag    21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

Let's get fancier. Let's get all the Mercedes without having to type them out. For those totally new to programming, this may be a little complicated. We will come back to the concepts illustrated here, so don't worry if it feels hard right now. But hopefully this illustrates the flexibility you have for manipulating a data frame.

```r
# First I will get the first four characters of the row names using substr()
sub4 <- substr(rownames(mtcars),1,4)

# Next, I will create a logical vector that is equal to TRUE when sub4 is equal to "Merc"
```

```r
is.Merc <- sub4=="Merc"

# Let's take a look at is.Merc
print("is.Merc:")
```

```
## [1] "is.Merc:"
```

```r
is.Merc
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
## [13]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```r
# Finally, let's make a new data frame that is just the rows for Mercedes cars
mtcars.Mercedes <- mtcars[is.Merc,]

# Let's look at this new data frame
mtcars.Mercedes
```

```
##              mpg cyl  disp  hp drat   wt qsec vs am gear carb
## Merc 240D   24.4   4 146.7  62 3.69 3.19 20.0  1  0    4    2
## Merc 230    22.8   4 140.8  95 3.92 3.15 22.9  1  0    4    2
## Merc 280    19.2   6 167.6 123 3.92 3.44 18.3  1  0    4    4
## Merc 280C   17.8   6 167.6 123 3.92 3.44 18.9  1  0    4    4
## Merc 450SE  16.4   8 275.8 180 3.07 4.07 17.4  0  0    3    3
## Merc 450SL  17.3   8 275.8 180 3.07 3.73 17.6  0  0    3    3
## Merc 450SLC 15.2   8 275.8 180 3.07 3.78 18.0  0  0    3    3
```

Also, using brackets, we can delete columns or rows from a data frame (or matrix). For example, let's drop the columns named `drat` and `am`. Then let's drop the `Datsun 710`. Here I introduce the `which` function. It returns the column numbers (or row numbers) for which a statement is true. I also introduce `%in%` which evaluates overlap between two vectors.

```r
# Here I designate the column numbers I want to drop
drop.cols <- which(colnames(mtcars) %in% c("drat","am"))

# Here I drop those columns
mtcars.reduced.columns <- mtcars[,-drop.cols]

# Here I find the row number for the Datsun 710 (note == since not using a vector)
drop.rows <- which(rownames(mtcars) == "Datsun 710")

# Here I drop the row
mtcars.reduced.columns.rows <- mtcars.reduced.columns[-drop.rows,]

# I could also drop both the rows and columns at the same time
mtcars.reduced.columns.rows <- mtcars[-drop.rows,-drop.cols]

# Let's look at the first 6 rows of the new data frame
head(mtcars.reduced.columns.rows)
```

```
##                    mpg cyl disp  hp    wt  qsec vs gear carb
## Mazda RX4         21.0   6  160 110 2.620 16.46  0    4    4
## Mazda RX4 Wag     21.0   6  160 110 2.875 17.02  0    4    4
## Hornet 4 Drive    21.4   6  258 110 3.215 19.44  1    3    1
## Hornet Sportabout 18.7   8  360 175 3.440 17.02  0    3    2
## Valiant           18.1   6  225 105 3.460 20.22  1    3    1
## Duster 360        14.3   8  360 245 3.570 15.84  0    3    4
```

NOTE GOOD CODING PRACTICE: When making modifications to a data frame, it is often wise to create new data frames to hold the revisions, rather than making modifications and then keeping the same object name. This is especially true when subtracting or renaming things or changing the content of the data - such that code run on different versions of a data frame would lead to different results. This practice makes it easier to check your code, keep track of changes, and avoid errors.

2. **With data frames, we can also use "data.frame.name$" to refer to columns/variables.**

Here we create a vector that is equal to the variable/column `mpg`.

```r
# The object mpg is being assigned the column names "mpg"
mpg <- mtcars$mpg

# Let's look at the mean value of mpg
mean(mpg)
```

```
## [1] 20.09062
```

We can also use this approach to add columns to a data frame. In the example below, we create a new column `kpg` that converts miles per gallon (`mpg`) to kilometers per gallon.

```r
# note since we are adding something new, rather than changing or deleting, it is fine to keep the same
mtcars$kpg <- mtcars$mpg * 1.6
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb   kpg
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4 33.60
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4 33.60
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1 36.48
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1 34.24
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2 29.92
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1 28.96
```

Reading and writing data files / data frames

.RData (or .Rda) files

- .RData files can store multiple R objects. Sometimes data files are stored as .rds files, which only store a single object.
- To load an .RData file, use the load() function. This function loads objects stored in the .RData file into your R environment.
- After loading, the objects contained in the .RData file will be available in your R environment with their original names.

```r
load("data/SleepStudy.Rda")
```

After making modifications to your data, you can save it as an `.RData` or `.Rda` file using the save() function. And to save a `.rds` file, saveRDS(). For example:

```r
# To save an .Rdata or .Rda file
#save(my.object, file = "path/output.Rda")

# to save an .rds file
#saveRDS(my_data, "path/output.rds")
```

.csv files or .txt files

A common way to import external data into R is from a CSV file. Use the read.csv() function.

```
#data <- read.csv("path/file.csv")
```

For plain text files, use read.table() or readLines():

```
#data <- read.table("path/file.txt", header = TRUE)

#lines <- readLines("path/file.txt")
```

You can write out a `.csv` using the write.csv() function and a `.txt` file using write.table().

```
# write data frame to .csv file
# write.csv(my.data, "path/output.csv")

# write to a .text file
# write.table(my.data, "path/output.txt", row.names = FALSE)
```

Tips

- File Paths: File paths can be absolute or relative. Absolute paths specify the exact location on your computer. Relative paths are relative to your current working directory in R. We set up R Studio so that it should be easy to use relative paths!
- Working Directory: Use getwd() to find your current working directory and setwd("path/to/directory") to change it. But we set up R Studio so that if you .Rproj is loaded, you current working directory is set.
- Viewing Data: After reading a file, use head(data) to view the first few lines of your data frame.
- Handling Errors: If R can't read your file, check for typos in the file path and ensure the file format is correct.