MTH 3240 R Notes 6

6 Data Frames

6.1 Creating and Viewing Data Frames

- ***Data frames*** are two-dimensional, like matrices, but they can be *heterogeneous* (a mix of some *categorical* `"character"` columns and some *numerical* ones).

- Each each row corresponds to an individual (person, place, thing, etc.), and each column contains the value of a ***variable*** observed (or measured) on that individual. An entire row is called an ***observation***.

- For example, the following data set (from a study of eight mice) could be stored as a *data frame* in R:

Mice Data Set

Color	Weight	Length
white	23	3.8
grey	21	3.7
black	18	3.0
brown	26	3.4
black	25	3.4
white	22	3.1
black	26	3.5
white	19	3.2

  Each row corresponds to a mouse, and each column a variable observed or measured on the mouse. Note that the data set has a mix of `"character"` and numerical columns, and therefore *couldn't* be stored as a matrix.

6.1.1 Creating Data Frames Using `data.frame()`

- For small data sets, we can create a data frame on the R command line using:

```
data.frame()       # Create a data frame from a set of vectors of the
                   # same length
```

  (We'll see later how to create data frames by reading larger data sets from a file.)

- In addition, several functions let us view various aspects of a data frame:

```
head(); tail()   # Prints the first (or last) six rows of a data frame
nrow(); ncol()   # Indicates the number of rows (or columns)
str()            # Gives the structure of a data frame
is.data.frame()  # Indicates whether or not an object is a data frame
```

- Consider again the **mice data set** from above. After creating vectors containing the data:

```
col <- c("white", "grey", "black", "brown", "black", "white", "black", "white")
wt <- c(23, 21, 12, 26, 25, 22, 26, 19)
len <- c(3.8, 3.7, 3.0, 3.4, 3.4, 3.1, 3.5, 3.2)
```

we create a data frame named `mice.data` by typing:

```
mice.data <- data.frame(Color = col, Weight = wt, Length = len,
                        stringsAsFactors = FALSE)
```

We can check that the data frame was created properly by typing its name:

```
mice.data

##   Color Weight Length
## 1 white     23    3.8
## 2  grey     21    3.7
## 3 black     12    3.0
## 4 brown     26    3.4
## 5 black     25    3.4
## 6 white     22    3.1
## 7 black     26    3.5
## 8 white     19    3.2
```

and we can make sure it is a data frame by typing:

```
is.data.frame(mice.data)

## [1] TRUE
```

Specifying `stringsAsFactors = FALSE` in `data.frame()` indicates that we *don't* want the `"character"` vector `col` to be converted to a so-called *factor* when the data frame is created. (The default for the argument `stringsAsFactors` is `TRUE`.) We'll discuss *factors* later.

- The functions `head()` and `tail()` print just the first and last six rows, respectively:

```
head(mice.data)
```

```
##   Color Weight Length
## 1 white     23    3.8
## 2  grey     21    3.7
## 3 black     12    3.0
## 4 brown     26    3.4
## 5 black     25    3.4
## 6 white     22    3.1
```

They're more useful with large data sets.

- To find out how many rows a data frame has, type:

```
nrow(mice.data)

## [1] 8
```

To find out how many columns it has, use `ncol()`.

- To look at the "structure" of `mice.data`, type:

```
str(mice.data)

##   data.frame: 8 obs. of  3 variables:
##  $ Color : chr  "white" "grey" "black" "brown" ...
##  $ Weight: num  23 21 12 26 25 22 26 19
##  $ Length: num  3.8 3.7 3 3.4 3.4 3.1 3.5 3.2
```

This indicates that there are eight observations of three variables, `Color`, a `"character"` vector, and `Weight` and `Length`, both numeric vectors.

6.1.2 Creating Data Frames by Reading Data from a File Using `read.table()`

- We can create a data frame by reading the data from a file using:

```
read.table()        # Read data from a text (.txt) file into a data
                    # frame
read.csv()          # Read data from a 'comma separated value' (.csv)
                    # file or a text (.txt) file into a data frame
file.choose()       # Opens a dialog box for choosing a file
```

- Suppose we have a text (.txt) file, **C:\Users\MyName\Documents\mice.txt**, that contains the **mice data set**. The contents of the text file would look like this:

Color	Weight	Length
white	23	3.8
grey	21	3.7
black	18	3.0
brown	26	3.4
black	25	3.4
white	22	3.1
black	26	3.5
white	19	3.2

- We read it into a data frame named `mice` by typing:

  ```
  # We could also use read.csv() here instead of read.table():
  mice.data <- read.table("C:/Users/MyName/Documents/mice.txt",
                          header = TRUE, stringsAsFactors = FALSE)
  ```

  It's always a good idea to check that the data were read in correctly:

  ```
  mice.data

  ##   Color Weight Length
  ## 1 white     23    3.8
  ## 2  grey     21    3.7
  ## 3 black     18    3.0
  ## 4 brown     26    3.4
  ## 5 black     25    3.4
  ## 6 white     22    3.1
  ## 7 black     26    3.5
  ## 8 white     19    3.2
  ```

  Above, we could also have used `read.csv()` instead of `read.table()`.

  `read.table()` and `read.csv()` are the *same function*, but have different default settings for some of the arguments.

- Here are some comments about using `read.table()` and `read.csv()`:
  - The usual back slashes (\) are written as forward slashes (/) in `read.table()` and `read.csv()`.
  - Specifying `header = TRUE` is used to indicate that the first row of the text file contains the variable names.
  - You can use either single (`'`) or double (`"`) quotations when specifying the location and name of the file.
  - By default, R recognizes one or more white spaces, tabs, or newline characters as separators of data values in the input file. Other separators can be specified via the `sep` argument to `read.table()` and `read.csv()`.
  - As before, specifying `stringsAsFactors = FALSE` indicates that we *don't* want the `"character"` column (Color) to be converted to a *factor*.

- To read data from an **Excel** file, you can either:
  - Save the Excel file as a tab delimited text (.txt) file, and then use `read.table()`, or
  - Save the Excel file as a 'comma separated value' (.csv) file, and then read it into R using `read.csv()`.

  (There are also more specialized functions for reading in Excel files, but we won't cover them.)

- Using `file.choose()` makes it easier to specify the file location and name. It opens a dialog box for choosing the file.

  For example, we could read **mice data** from the text (.txt) file into a data frame named `mice` by first typing:

```
my.file <- file.choose()
```

and selecting the **mice.txt** file in the dialog box, then typing:

```
# We could also use read.csv() here instead of read.table():
mice.data <- read.table(my.file,
                        header = TRUE,
                        stringsAsFactors = FALSE)
```

Section 6.1 Exercises

Exercise 1

a) Here's the **mice data set** as three vectors:

```
col <- c("white", "grey", "black", "brown", "black", "white", "black",
         "white")
wt <- c(23, 21, 12, 26, 25, 22, 26, 19)
len <- c(3.8, 3.7, 3.0, 3.4, 3.4, 3.1, 3.5, 3.2)
```

After creating the vectors, write a command involving `data.frame()` that creates a data frame containing the data. Make sure the column names are `Color`, `Weight`, and `Length`.

Check that you created the data frame correctly by typing its name on the command line by typing:

```
mice.data
```

Report on what you see.

b) Before proceeding, remove the data frame you just created from your Workspace, for example by typing:

```
rm(mice.data)
```

and double check that it's no longer there by typing:

```
ls()                      # Or you could use objects().
```

The file **mice.txt** (on the course website) contains the **mice data set**. After saving the file onto your computer, type the following:

```
my.file <- file.choose()
```

then in the dialog box, select the **mice.txt** file (that you saved). Now type:

```
my.file
```

What information is stored in `my.file`?

c) Now read the data from the **mice.txt** file into R by typing:

```
mice.data <- read.table(my.file,
                        header = TRUE,
                        stringsAsFactors = FALSE)
```

Check that the data were read in correctly by typing:

```
mice.data
```

Report on what you see.

6.2 Accessing and Replacing Elements, Rows, or Columns of a Data Frame

- Data frames have features of both *matrices and lists*. To extract a specific value, row, or column, we use:

```
[ , ]      # Access data frame elements, rows, or columns via their
           # row and column indices (separated by a comma)
$          # Access a data frame variable (column) by specifying its
           # name
```

6.2.1 Accessing Rows and Columns Using `[ ]`

- As with matrices, we can access a specific element, row, or column of a data frame using single square brackets `[ ]`. For example, the value in the 3rd row and 2nd column of `mice.data` is:

  ```
  mice.data[3, 2]

  ## [1] 18
  ```

  The entire 3rd row is accessed via:

  ```
  mice.data[3, ]

  ##   Color Weight Length
  ## 3 black     18      3
  ```

  and the entire 2nd column could be accessed via `mice.data[, 2]`.

- We can also use square brackets and the assignment operator to *replace* a value in a data frame, for example:

  ```
  mice.data[3, 2] <- 12
  ```

6.2.2 Accessing and Replacing Columns Using `$`

- In fact, **data frames are *lists***:

  ```
  is.data.frame(mice.data)

  ## [1] TRUE

  is.list(mice.data)

  ## [1] TRUE
  ```

  The list elements are the columns of the data frame.

- Therefore, the list operator `$` can also be used to access columns of a data frame:

  ```
  mice.data$Color           # Access a variable (column) by name

  ## [1] "white" "grey"  "black" "brown" "black" "white" "black" "white"
  ```

- We *replace* a column using the assignment operator. For example:

```
mice.data$Color <- c("blue", "white", "orange", "pink", "magenta", "blue", "orange", "blue")
```

Above, we could also have replaced the `Color` column using `mice.data[, 1] <-` , in which case R would've chosen a name for the variable in the data frame

6.2.3 Adding a New Column Using `[ ]` or `$`

- We can add a column to a data frame using `[ ]` or `$` and the assignment operator. For example, here are the mice body fats (in grams):

```
Bodyfat <- c(2.5, 2.1, 3.1, 3.0, 2.7, 2.6, 1.8, 2.0)
```

To add these as a column to the `mice.data`, either of the following will work:

```
mice.data$Fat <- Bodyfat          # The variable in mice.data will be named Fat
mice.data[ , 4] <- Bodyfat        # R will choose a name for the variable in mice.data
```

As per the comments, only the first method above allows you to choose a name for the new column. You'd have to add a name later (using `names()`) if you used the second one.

Section 6.2 Exercises

Exercise 2 Consider the following data set on nine people:

Status	Age	Education
Married	36	HS Diploma
Single	33	Bachelor of Arts
Single	21	Bachelor of Science
Married	29	Bachelor of Science
Single	19	HS Diploma
Married	35	Bachelor of Arts
Married	39	Master of Science
Single	28	HS Diploma
Single	21	HS Diploma

Here are vectors containing the data:

```
status <- c("Married", "Single", "Single", "Married", "Single",
              "Married", "Married", "Single", "Single")
age <- c(36, 33, 21, 29, 19, 35, 39, 28, 21)
educ <- c("HS Diploma", "Bachelor of Arts", "Bachelor of Science",
            "Bachelor of Science", "HS Diploma", "Bachelor of Arts",
            "Master of Science", "HS Diploma", "HS Diploma")
```

and here's a command that will create a data frame containing the data:

```
my.data <- data.frame(Status = status, Age = age, Education = educ)
```

a) Write a command involving single square brackets `[ ]` that returns the age of the person in the 6th row.

b) Write a command using single square brackets `[ ]` that returns the entire 3rd row of the data frame.

c) Write two different commands that return the entire 2nd column of the data frame:

- Using single square brackets `[ ]`.
- Using the dollar sign operator `$`.

d) The nine people have each aged one year since the data were collected. Here's a vector containing their current ages:

```
age2 <- c(37, 34, 22, 30, 20, 36, 40, 29, 22)
```

What does the following command do?

```
my.data$Age2 <- age2
```

6.3 Viewing and Changing Variable Names in a Data Frame

- We can get or change the names of the columns of a data frame using:

```
names()           # Get or assign the names of the variables in a
                  # data frame
```

- For example, to *view* the column names in the `mice.data` data frame (from above), we type:

```
names(mice.data)

## [1] "Color"  "Weight" "Length"
```

- To *change* the column names, we could use:

```
names(mice.data) <- c("Col", "Wt", "Len")
```

```
names(mice.data)

## [1] "Col" "Wt"  "Len"
```

- To change just one of the column names, such as the 3rd one, we could use:

```
names(mice.data)[3] <- "Len"
```

Section 6.3 Exercises

Exercise 3 Create the following data frame:

```
x <- data.frame(A = 1:5, B = 6:10, C = c("a", "b", "c", "d", "e"))
```

a) Guess what the following command will return, then check your answer:

```
names(x)
```

b) Guess what the following will do, then check your answer:

```
names(x) <- c("AA", "BB","CC")
```

c) Now write a command that changes the name of the 3rd column of `x` to `"DD"`.

6.4 Rearranging the Rows or Columns of a Data Frame

6.4.1 Rearranging the Rows or Columns

- We rearrange (permute) the rows (or columns) of a data frame just as is done for a matrix, using a vector of indices with the desired permutation before (or after) a comma in square brackets `[ ]`.

  For example here are the **mice data** again:

```
mice.data

##     Col Wt Len
## 1 white 23 3.8
## 2  grey 21 3.7
## 3 black 12 3.0
## 4 brown 26 3.4
## 5 black 25 3.4
## 6 white 22 3.1
## 7 black 26 3.5
## 8 white 19 3.2
```

  To reorder the columns as `Wt`, `Len`, `Col`, we type:

```
mice.data[ , c(3, 2, 1)]                # Rearranges the columns

##   Len Wt   Col
## 1 3.8 23 white
## 2 3.7 21  grey
## 3 3.0 12 black
## 4 3.4 26 brown
## 5 3.4 25 black
## 6 3.1 22 white
## 7 3.5 26 black
## 8 3.2 19 white
```

and to put the rows in reverse order, we'd type:

```
mice.data[8:1, ]                   # Rearranges the rows

##     Col Wt Len
## 8 white 19 3.2
## 7 black 26 3.5
## 6 white 22 3.1
## 5 black 25 3.4
## 4 brown 26 3.4
## 3 black 12 3.0
## 2  grey 21 3.7
## 1 white 23 3.8
```

6.4.2 Sorting Rows by the Values of One Variable

- We can sort the rows of a data frame according to the values of one of its variables (columns) using square brackets `[ ]` and the `order()` function.

  For example, to order the *rows* of `mice.data` by the `Wt`s of the mice, we type:

```
mice.data[order(mice.data$Wt), ]

##     Col Wt Len
## 3 black 12 3.0
## 8 white 19 3.2
## 2  grey 21 3.7
## 6 white 22 3.1
## 1 white 23 3.8
## 5 black 25 3.4
## 4 brown 26 3.4
## 7 black 26 3.5
```

  Notice that the rows have been sorted according increasing `Wt`s.

- To see how the above command works, note first that `order()` takes a vector argument `x` and returns a set of indices that will put the elements of `x` in increasing order:

  ```
  x <- c(7, 9, 5)
  order(x)                        # Returns the indices that will sort x

  ## [1] 3 1 2
  ```

  This says that the 3rd element of `x` is the smallest, the 1st element is the second smallest, and the 2nd element is the largest. Thus to sort `x`, we'd type:

  ```
  x[c(3, 1, 2)]

  ## [1] 5 7 9
  ```

  or just:

  ```
  x[order(x)]                     # This does the same thing as sort(x)

  ## [1] 5 7 9
  ```

  Now here's the set of indices that will order the `Wts` of the mice from `mice.data` from smallest to largest:

  ```
  order(mice.data$Wt)

  ## [1] 3 8 2 6 1 5 4 7
  ```

  Thus to sort the rows of `mice.data`, we'd type:

  ```
  mice.data[c(3, 8, 2, 6, 1, 5, 4, 7), ]
  ```

  or just

  ```
  mice.data[order(mice.data$Wt), ]
  ```

  as was used previously.

Section 6.4 Exercises

Exercise 4 Here's a small data frame:

```
x <- data.frame(x1 = c("a", "b", "c"), x2 = c(1, 2, 3))
x

##   x1 x2
## 1  a  1
## 2  b  2
## 3  c  3
```

a) Guess what the following command will do, then check your answer.

```
x[, c(2, 1)]
```

b) Guess what the following command will do, then check your answer.

```
x[c(3, 1, 2), ]
```

Exercise 5 Here's another small data frame:

```
x <- data.frame(x1 = c(6, 4, 5), x2 = c("h", "g", "f"),
                stringsAsFactors = FALSE)
x
```

```
##   x1 x2
## 1  6  h
## 2  4  g
## 3  5  f
```

What does the following command do? Check your answer:

```
x[order(x$x1), ]
```

6.5 Filtering on Data Frames

- Recall that **filtering** means extracting a subset of rows that satisfy some condition. We can filter rows from a data frame using either square brackets `[ ]` or `subset()`.

6.5.1 Filtering Rows Using Square Brackets `[ ]`

- As an example, to extract from `mice.data` the rows corresponding to *black mice* using square brackets `[ ]`, we type:

```
mice.data[mice.data$Col == "black", ]     # mice.data£Col=="black" is a "log-
ical" vector
```

```
##      Col Wt Len
## 3 black 12 3.0
## 5 black 25 3.4
## 7 black 26 3.5
```

Note that `mice.data$Col=="black"` is a `"logical"` vector.

6.5.2 Filtering Rows Using `subset()`

- `subset()` takes two main arguments, a data frame, `x`, and a condition to be met, `subset`, and then extracts from `x` the rows for which the condition is met.
- For example, to extract the rows of `mice.data` the rows corresponding to *black mice*, type:

```
subset(mice.data, subset = Col == "black")

##      Col Wt Len
## 3 black 12 3.0
## 5 black 25 3.4
## 7 black 26 3.5
```

Notice that there's no need to use the dollar sign operator `$` with `Col` when `subset()` is used.

Section 6.5 Exercises

Exercise 6 The built-in R data frame `warpbreaks` contains data from a study of the strength of yarn used in weaving. There are three variables in the data set:

`breaks`	The number of breaks per loom, where a loom corresponds to a fixed length of yarn.
`wool`	The type of wool (A or B)
`tension`	The level of tension (L, M, H)

To look at the data, type:

```
warpbreaks
```

If you want to read about it, look at its help page by typing:

```
? warpbreaks
```

a) Guess what the following command will do, then check your answer:

```
warpbreaks[warpbreaks$tension == "M", ]
```

b) Now guess what the following command will do, then check your answer:

```
subset(warpbreaks, subset = tension == "M")
```

6.6 More on the Treatment of `NA`s

- Recall that `NA` is used in R to represent a data value that's supposed to be present but is missing.

  If a data set contains `NA`s, we may want to exclude from the data analysis any rows for which one or more values are `NA`. The following function is useful in this regard:

```
complete.cases()      # Checks whether or not each row of a data frame
                      # is complete (i.e. doesn't contain NAs), and
                      # returns a "logical" vector indicating whether
                      # each row is complete.
```

- `complete.cases()` takes a data frame as its main argument, and returns a `"logical"` vector whose elements are `TRUE` if the corresponding row of the data frame is "complete" (doesn't contain any `NA`s) and `FALSE` otherwise.

  Consider, for example, the following data frame:

```
cars

##           Make  Model Year CityMPG HighwayMPG
## 1         Ford  Focus 2012      17         25
## 2       Toyota  Prius 2013      51         NA
## 3         Ford Fusion 2014      22         31
## 4    Chevrolet   Volt 2015      NA         NA
## 5        Honda Accord 2011      23         33
## 6   Volkswagen Beetle 2012      22         31
## 7    Chevrolet Impala 2015      22         31
## 8        Tesla ModelS 2014      NA         NA
## 9        Honda  Civic 2011      26         34
## 10       Honda  S2000 2005      17         23
## 11      Toyota  Camry 2013      25         35
```

  To determine which rows are "complete", type:

```
complete.cases(cars)

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

  This says that all of the rows *except* the 2nd, 4th, and 8th are "complete".

  We use the `"logical"` vector returned by `complete.cases()` to return a data frame's "complete" rows as follows:

```
subset(cars, subset = complete.cases(cars))

##           Make  Model Year CityMPG HighwayMPG
## 1         Ford  Focus 2012      17         25
## 3         Ford Fusion 2014      22         31
## 5        Honda Accord 2011      23         33
## 6  Volkswagen Beetle 2012      22         31
## 7   Chevrolet Impala 2015      22         31
## 9        Honda  Civic 2011      26         34
## 10       Honda  S2000 2005      17         23
## 11      Toyota  Camry 2013      25         35
```

We could also accomplish the same thing using square brackets `[ ]` by typing:

```
cars[complete.cases(cars), ]
```

Section 6.6 Exercises

Exercise 7 Here's a data frame:

```
x <- data.frame(x1 = c(8, 2, 6, NA, 9, 4, 3),
                x2 = c("u", "r", "s", "v", "c", "t", "w"),
                x3 = c(2, NA, 4, 7, 7, NA, 1))
x
```

```
##   x1 x2 x3
## 1  8  u  2
## 2  2  r NA
## 3  6  s  4
## 4 NA  v  7
## 5  9  c  7
## 6  4  t NA
## 7  3  w  1
```

a) Guess what the following command will return, then check your answer:

```
complete.cases(x)
```

b) Now write a command using `subset()` (or square brackets `[ ]`) that returns just the rows of `x` that are ”complete” (i.e. that don’t contain `NA`s).