Data Wrangling: February 7, 2018

Melinda Higgins

February 7, 2018

Data Wrangling: February 7, 2018

In today's class we will cover the various data (or "object") structures in `R`. We will cover the following objects:

- scalar (which are really vectors of length 1)
- vector
- matrices (and "arrays")
- data frame
- list
- factors

NOTE: To learn more about data types in R see:

- Quick-R Pages on datatypes http://www.statmethods.net/input/datatypes.html
- Jenny Bryan's slides on R Objects https://speakerdeck.com/jennybc/simple-view-of-r-objects

We will also cover the following useful functions for checking/reviewing the data types of objects in R:

- `summary()`
  - summary is a generic function used to produce result summaries of the results of various model fitting functions. The function invokes particular methods which depend on the class of the first argument.
- `str()`
  - Compactly display the internal structure of an R object, a diagnostic function and an alternative to summary (and to some extent, dput). Ideally, only one line for each 'basic' structure is displayed. It is especially well suited to compactly display the (abbreviated) contents of (possibly nested) lists. The idea is to give reasonable output for any R object. It calls args for (non-primitive) function objects.
- `class()`
  - R possesses a simple generic function mechanism which can be used for an object-oriented style of programming. Method dispatch takes place based on the class of the first argument to the generic function.
- `mode()`
  - Get or set the type or storage mode of an object.
- `typeof()`
  - typeof determines the (R internal) type or storage mode of any object
- `attributes()`
  - These functions access an object's attributes. The first form below returns the object's attribute list. The replacement forms uses the list on the right-hand side of the assignment as the object's attributes (if appropriate).

NOTE: Many times if you get an error trying to run a function in `R` it will be because the object you have put into the function is of the wrong class. You'll get something warning you that there is a problem or that you have a class type mismatch.

EXAMPLE: Read the help pages for the `lm()` function which is used to fit a linear model. the usage is

```
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
   singular.ok = TRUE, contrasts = NULL, offset, ...)
```

where:

- the 1st argument `formula` is assumed to an object of class "formula" - which we'll discuss later this semester
- the second argument `data` is assumed to a data frame or list
- the 3rd argument `subset` is assumed to be a vector . . . and so on . . . ALWAYS read the help pages for the function and see what types of objects it wants

Let's create some data/objects

Create 5 scalar values.

```
a <- 3
b <- 1
c <- 5
d <- 4.5
e <- 6.234
```

Create some objects

We could use these individual values to create a single `vector` containing these 5 values. To do this we'll use the `c()` "combine values" function. We can either do it using the object names `"a" "b" "c" "d" "e"` or we can simply use the values themselves.

```
v1 <- c(a,b,c,d,e)
v2 <- c(3,1,5,4.5,6.234)
```

Let's make a couple more vectors with 5 elements. Let's make one with characters/letters and another one with TRUE/FALSE (or T/F) logical values and a third with only whole numbers.

```
v.char <- c("blue","green","red","yellow","blue")
v.log <- c(T,F,F,T,F)
v.int <- c(2002,2004,2006,2008,2010)
```

Look at the properties of the objects

Look at the class of each object

```r
class(v1)
```

```
## [1] "numeric"
```

```r
class(v2)
```

```
## [1] "numeric"
```

```r
class(v.char)
```

```
## [1] "character"
```

```r
class(v.log)
```

```
## [1] "logical"
```

```r
class(v.int)
```

```
## [1] "numeric"
```

Look at the mode

```r
mode(v1)
```

```
## [1] "numeric"
```

```r
mode(v2)
```

```
## [1] "numeric"
```

```r
mode(v.char)
```

```
## [1] "character"
```

```r
mode(v.log)
```

```
## [1] "logical"
```

```r
mode(v.int)
```

```
## [1] "numeric"
```

Look at the typeof each object

```r
typeof(v1)
```

```
## [1] "double"
```

```r
typeof(v2)
```

```
## [1] "double"
```

```r
typeof(v.char)
```

```
## [1] "character"
```

```r
typeof(v.log)
```

```
## [1] "logical"
```

```r
typeof(v.int)
```

```
## [1] "double"
```

NOTE: Notice that `v.int` is a numeric vector not an integer. We can force this to a true integer vector using `is.integer()` to test to see if `R` thinks is it is an integer vector. If not, then we can use `as.integer()` to force it.

```r
is.integer(v.int)
```

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

```r
v.int2 <- as.integer(v.int)
is.integer(v.int2)
```

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

Object types and changes

Here are some useful functions for checking your object types and for possibly changing them as needed.

- `is.character()`
- `is.numeric()`
- `is.integer()`
- `is.data.frame()`
- `is.double()`
- `is.list()`
- `is.factor()`
- and there are more - just look in help for the various `is.xxx()` functions.

Most of these have associated `as.xx()` functions to help you move back and forth between object types ~ *sometimes*.

- `as.character()`
- `as.numeric()`
- `as.integer()`
- `as.data.frame()`
- `as.double()`
- `as.list()`
- `as.factor()`
- and there are more - just look in help for the various `as.xxx()` functions.

Make a `data.frame`

Thus far, all of our vectors are of the same length, but they are of different type. The best data object to hold (a) vectors of the same length and (b) of different data types is a `data.frame`. The `data.frame` object is the best form of **TIDY** data where each ROW is a single CASE (or SUBJECT) and each COLUMN is a feature, measurement, or piece of information about that case (i.e. the COLUMNS are the VARIABLES).

Let's combine v2, v.char, v.int2, and v.log into a `data.frame`. The easiest way to do this is using the `data.frame()` function. Then let's run

- `str()` to see the structure listing
- `summary()` to see what summary stats we get for each column since they are different data types
- let's also look at the `class()`,
- `mode()`,
- `typeof()` and
- `attributes` for the newly created `df` `data.frame` object

```
# create a data.frame object called "df"
# combining 4 vectors v2, v.char, v.int2, v.log
df <- data.frame(v2,v.char,v.int2,v.log)

# look at the structure
str(df)
```

```
## 'data.frame':    5 obs. of  4 variables:
##  $ v2    : num  3 1 5 4.5 6.23
##  $ v.char: Factor w/ 4 levels "blue","green",..: 1 2 3 4 1
##  $ v.int2: int  2002 2004 2006 2008 2010
##  $ v.log : logi  TRUE FALSE FALSE TRUE FALSE
```

```
# look at the other properties of "df"
class(df)
```

```
## [1] "data.frame"
```

```
mode(df)
```

```
## [1] "list"
```

```r
typeof(df)
```

```
## [1] "list"
```

```r
attributes(df)
```

```
## $names
## [1] "v2"     "v.char" "v.int2" "v.log"
##
## $row.names
## [1] 1 2 3 4 5
##
## $class
## [1] "data.frame"
```

Attributes of a `data.frame`

Notice that when we ran `attributes(df)` there were 3 pieces of information provided:

- `$names`
- `$row.names`
- and `$class`

The `names` attributes is really helpful for labeling and selecting specific COLUMNS or VARIABLES in our new dataset "df". There is a function `names()` that is useful for (a) finding out what the column/variables names are and for (b) changing the variable names if we need to.

Right now the column names are not helpful. The names simply reflect the previous vector names.

```r
names(df)
```

```
## [1] "v2"     "v.char" "v.int2" "v.log"
```

Let's change the names to something more interesting.

- for "v2" we'll change that to "avgvisit" (hypothetical average number of visits to somewhere)
- for "v.char" change that to "color" (hypothetical color categories for plotting later)
- for "v.int2" change to "year" (hypothetical year the data was collected)
- and for "v.log" change to "valid" (hypothetical indicator for whether the data is validated or not)

To do this we'll use the `c()` combine function and assemble these new labels to rename the current column names. Run the `names(df)` before applying the new names, then assign the new names, and run `names(df)` again to see/check that the column names have been updated.

```r
# see the original variable/column names
names(df)
```

```
## [1] "v2"     "v.char" "v.int2" "v.log"
```

```r
# assign the new variable/column names
names(df) <- c("avgvisit","color","year","valid")

# see that the variable/column names have updated
names(df)
```

```
## [1] "avgvisit" "color"    "year"     "valid"
```

View/Print the dataset (as a table)

Since this is such a small dataset with 5 ROWS and 4 COLUMNS we can easily "view" it by printing it in a table. For this we'll use the `kable()` function from the `knitr` package. To call a specific function in a specific package, you list the package first followed by 2 colons `::` and then the function.

NOTE: The `kable()` function makes pretty good tables in HTML, DOCX and PDF formats for objects that are `data.frame` or a `matrix`. We'll learn more about the `kable()` function throughout the semester... In the example below, I also added a `caption` for the table.

```r
knitr::kable(df,
             caption="View the 'df' object")
```

Table 1: View the 'df' object

avgvisit	color	year	valid
3.000	blue	2002	TRUE
1.000	green	2004	FALSE
5.000	red	2006	FALSE
4.500	yellow	2008	TRUE
6.234	blue	2010	FALSE

Worked Example from the UCI Data Repository

The following dataset comes from the UCI Data Repository. The dataset we'll use is the Contraceptive Method Choice dataset. The information on this dataset is provided at http://archive.ics.uci.edu/ml/datasets/Contraceptive+Method+Choice. If you click on the "Data Folder" you can download the RAW data `cmc.data` which is a comma delimited format dataset (i.e. it is a CSV formatted file) and the description of the data included, the variable names and associated codes for the values included which is in the `cmc.names` file. See "Data Folder"" at http://archive.ics.uci.edu/ml/machine-learning-databases/cmc/

Read-in data

NOTE: Download the 2 files from the UCI Data Repository for the Contraceptive Method Choice and put them in the directory where you have this RMD `rmarkdown` file.

```r
# load the tidyverse package(s)
library(tidyverse)
```

```
## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr

## Conflicts with tidy packages ----------------------------------------------

## filter(): dplyr, stats
## lag():    dplyr, stats
```

```
# read in the comma delimited (CSV) formatted dataset
# **NOTE**: This dataset does NOT have the column
# names as the 1st row of the file. We will assign the
# column names below.
cmc <- read_csv("cmc.data", col_names=FALSE)
```

```
## Parsed with column specification:
## cols(
##   X1 = col_integer(),
##   X2 = col_integer(),
##   X3 = col_integer(),
##   X4 = col_integer(),
##   X5 = col_integer(),
##   X6 = col_integer(),
##   X7 = col_integer(),
##   X8 = col_integer(),
##   X9 = col_integer(),
##   X10 = col_integer()
## )
```

Apply the codebook - variable names and coding used

Apply variable names to the 10 columns of data in `cmc`.

```
# the variable names before we change them
# Notice that the columns names are very generic
# X1, X2, ..., X10
names(cmc)
```

```
##  [1] "X1"  "X2"  "X3"  "X4"  "X5"  "X6"  "X7"  "X8"  "X9"  "X10"
```

```
# assign new variables names to the 10 columns
names(cmc) <- c("WifeAge", "WifeEd", "HusbEd", "NumChild",
                "WifeRel", "WifeWork", "HusbOcc", "SOLindex",
                "Media", "Contraceptive")

# see the updated column/variable names
names(cmc)
```

```
##  [1] "WifeAge"       "WifeEd"        "HusbEd"        "NumChild"
##  [5] "WifeRel"       "WifeWork"      "HusbOcc"       "SOLindex"
##  [9] "Media"         "Contraceptive"
```

The next code chunk is to add the labels for "factor" levels for some of the variables (i.e. we are creating factors).

WARNING: Notice I'm overwriting the variables and changing them from integers to factors which have different properties as you'll see below. If you want to keep the original integer variables, you could simply give the new facotr variable a new name. For example you could write

```
cmc$WifeEd.f <- factor(cmc$WifeEd,
                       levels = c(1,2,3,4),
                       labels = c("low","med low","med
```

and this would append a new column onto the `cmc` dataset that is the "factor" type version of Wife's Education. For now, use the code below to update all of the variables.

```r
# notice that Wife Education is currently of "Integer" class type
# see what you get from the summary() function
class(cmc$WifeEd)
```

```
## [1] "integer"
```

```r
summary(cmc$WifeEd)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   2.959   4.000   4.000
```

```r
# update Wife Education as a factor, assign the
# levels and the labels for each level
cmc$WifeEd <- factor(cmc$WifeEd,
                     levels = c(1,2,3,4),
                     labels = c("low","med low","med high","high"))

# repeat the above commands to see what changes
class(cmc$WifeEd)
```

```
## [1] "factor"
```

```r
summary(cmc$WifeEd)
```

```
##      low  med low med high     high
##      152      334      410      577
```

```r
# do the remaining variables
cmc$HusbEd <- factor(cmc$HusbEd,
                     levels = c(1,2,3,4),
                     labels = c("low","med low","med high","high"))

cmc$WifeRel <- factor(cmc$WifeRel,
                      levels = c(0,1),
                      labels = c("Non-Islam","Islam"))

# Note: The documentation does state that
# 0=yes and 1=no which seems incorrect...
cmc$WifeWork <- factor(cmc$WifeWork,
                       levels = c(0,1),
                       labels = c("Yes","No"))

cmc$HusbOcc <- factor(cmc$HusbOcc,
                      levels = c(1,2,3,4),
                      labels = c("1","2","3","4"))

cmc$SOLindex <- factor(cmc$SOLindex,
                       levels = c(1,2,3,4),
                       labels = c("low","med low","med high","high"))

cmc$Media <- factor(cmc$Media,
                    levels = c(0,1),
                    labels = c("Good","Not Good"))

cmc$Contraceptive <- factor(cmc$Contraceptive,
                            levels = c(1,2,3),
                            labels = c("No-use","Long-term","Short-term"))
```

Look at a subset of the data

```r
head(cmc)
```

```
## # A tibble: 6 x 10
##   WifeAge   WifeEd   HusbEd NumChild WifeRel WifeWork HusbOcc SOLindex
##     <int>   <fctr>   <fctr>    <int>  <fctr>   <fctr>  <fctr>   <fctr>
## 1      24  med low med high        3   Islam       No       2 med high
## 2      45      low med high       10   Islam       No       3     high
## 3      43  med low med high        7   Islam       No       3     high
## 4      42 med high  med low        9   Islam       No       3 med high
## 5      36 med high med high        8   Islam       No       3  med low
## 6      19     high     high        0   Islam       No       3 med high
## # ... with 2 more variables: Media <fctr>, Contraceptive <fctr>
```

Print this subset using `knitr::kable()`

```r
knitr::kable(head(cmc))
```

WifeAge	WifeEd	HusbEd	NumChild	WifeRel	WifeWork	HusbOcc	SOLindex	Media	Contraceptive
24	med low	med high	3	Islam	No	2	med high	Good	No-use
45	low	med high	10	Islam	No	3	high	Good	No-use
43	med low	med high	7	Islam	No	3	high	Good	No-use
42	med high	med low	9	Islam	No	3	med high	Good	No-use
36	med high	med high	8	Islam	No	3	med low	Good	No-use
19	high	high	0	Islam	No	3	med high	Good	No-use

Summarize the dataset

NOTICE that Wife's Age and Number of Children are now the only "numeric" "integer" variables - these are the only ones for which we get summary statistics. All the remaining variables are "factors" so we only get the frequencies for each category.

```
summary(cmc)
```

```
##     WifeAge           WifeEd         HusbEd       NumChild     
##  Min.   :16.00   low     :152   low     : 44   Min.   : 0.000  
##  1st Qu.:26.00   med low :334   med low :178   1st Qu.: 1.000  
##  Median :32.00   med high:410   med high:352   Median : 3.000  
##  Mean   :32.54   high    :577   high    :899   Mean   : 3.261  
##  3rd Qu.:39.00                                 3rd Qu.: 4.000  
##  Max.   :49.00                                 Max.   :16.000  
##       WifeRel     WifeWork   HusbOcc     SOLindex        Media     
##  Non-Islam: 220   Yes: 369   1:436   low     :129   Good    :1364  
##  Islam    :1253   No :1104   2:425   med low :229   Not Good: 109  
##                              3:585   med high:431                  
##                              4: 27   high    :684                  
##                                                                    
##                                                                    
##     Contraceptive
##  No-use    :629  
##  Long-term :333  
##  Short-term:511  
##                  
##                  
##                  
```

Computing stats on factors

Suppose you wanted to know the mean education level of the Huband's in this dataset. We can use the `as.numeric()` function to convert the variable and then run a `mean()` on it. We'll do more on facotrs later this semester.

```
mean(as.numeric(cmc$HusbEd))
```

```
## [1] 3.429735
```

Select various elements

```
# select the first 10 rows
cmc_first10 <- cmc[1:10,]
cmc_first10
```

```
## # A tibble: 10 x 10
##    WifeAge   WifeEd   HusbEd NumChild WifeRel WifeWork HusbOcc SOLindex
##      <int>   <fctr>   <fctr>    <int>  <fctr>   <fctr>  <fctr>   <fctr>
##  1      24  med low med high        3   Islam       No       2 med high
##  2      45      low med high       10   Islam       No       3     high
##  3      43  med low med high        7   Islam       No       3     high
##  4      42 med high  med low        9   Islam       No       3 med high
##  5      36 med high med high        8   Islam       No       3  med low
##  6      19     high     high        0   Islam       No       3 med high
##  7      38  med low med high        6   Islam       No       3  med low
##  8      21 med high med high        1   Islam      Yes       3  med low
##  9      27  med low med high        3   Islam       No       3     high
## 10      45      low      low        8   Islam       No       2  med low
## # ... with 2 more variables: Media <fctr>, Contraceptive <fctr>
```

```
# select the first 2 columns
cmc_cols12 <- cmc[,1:2]
cmc_cols12
```

```
## # A tibble: 1,473 x 2
##    WifeAge   WifeEd
##      <int>   <fctr>
##  1      24  med low
##  2      45      low
##  3      43  med low
##  4      42 med high
##  5      36 med high
##  6      19     high
##  7      38  med low
##  8      21 med high
##  9      27  med low
## 10      45      low
## # ... with 1,463 more rows
```

```
# select the value in 5th row, 6th column
cmc_row5col6 <- cmc[5,6]
cmc_row5col6
```

```
## # A tibble: 1 x 1
##   WifeWork
##     <fctr>
## 1       No
```

Select Rows

Use the following code to find the number of subjects (number of rows) where the wife's age is > 40.

```
# find the dimensions of the cmc dataset
dim(cmc)
```

```
## [1] 1473   10
```

```
# the 1st element is the number of rows, the 2nd
# is the number of columns - just look at number of rows
dim(cmc)[1]
```

```
## [1] 1473
```

```
# select only the rows where WifeAge > 40
cmc_wife_gt40 <- cmc[cmc$WifeAge>40,]

# get number of rows of this subset
dim(cmc_wife_gt40)[1]
```

```
## [1] 306
```

Select Columns

We can select columns by position or by name. Select the columns for the Wife variables, which are in columns 1, 2, 5, 6 - these variables are:

- WifeAge
- WifeEd
- WifeRel
- WifeWork

```
# select these columns by position
cmc_wifeOnly <- cmc[,c(1,2,5,6)]
head(cmc_wifeOnly)
```

```
## # A tibble: 6 x 4
##   WifeAge   WifeEd WifeRel WifeWork
##     <int>   <fctr>  <fctr>   <fctr>
## 1      24  med low   Islam       No
## 2      45      low   Islam       No
## 3      43  med low   Islam       No
## 4      42 med high   Islam       No
## 5      36 med high   Islam       No
## 6      19     high   Islam       No
```

```
# select these columns by variable names
keepvars <- c("WifeAge", "WifeEd", "WifeRel", "WifeWork")
cmc_wifeOnly <- cmc[,keepvars]
head(cmc_wifeOnly)
```

```
## # A tibble: 6 x 4
##   WifeAge   WifeEd WifeRel WifeWork
##     <int>   <fctr>  <fctr>   <fctr>
## 1      24  med low   Islam       No
## 2      45      low   Islam       No
## 3      43  med low   Islam       No
## 4      42 med high   Islam       No
## 5      36 med high   Islam       No
## 6      19     high   Islam       No
```

TIDY Data

We'll use the following functions as we go through this semester, for now, let's review the following package(s) and the whole `TIDYVERSE` which is very helpful for working with data in many different formats and structures.

- The "TIDYVERSE"
- "Tidy Data" paper by Hadley Wickham; Journal of Statistical Software; v.59 (2014)
  - **NOTE**: The original code and datasets can be reviewed at the Github repository for this paper at https://github.com/hadley/tidy-data.
- "R for Data Science" book by Hadley Wickham; O'Reilly Media Inc. (2017) - Part II on Data Wrangling