

# Data Wrangling: January 25, 2017

*Melinda Higgins*

*January 25, 2017*

## Data Wrangling: January 25, 2017

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

```
class(v1)
```

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

```
class(v2)
```

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

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

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

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

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

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

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

---

Look at the mode

```
mode(v1)

## [1] "numeric"
mode(v2)

## [1] "numeric"
mode(v.char)

## [1] "character"
mode(v.log)

## [1] "logical"
mode(v.int)

## [1] "numeric"
```

---

Look at the typeof each object

```
typeof(v1)

## [1] "double"
typeof(v2)

## [1] "double"
typeof(v.char)

## [1] "character"
typeof(v.log)

## [1] "logical"
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 it is an integer vector. If not, then we can use `as.integer()` to force it.

```
is.integer(v.int)

## [1] FALSE
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"
typeof(df)

## [1] "list"
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` attribute 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.

```
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.

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

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

# 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.

```
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.

```
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
```

```
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

*PLACEHOLDER* - code to be added for labeling the data and adding the coding levels and descriptions (i.e. creating the factors).

## Summarize the dataset

```
summary(cmc)
```

```
##           X1           X2           X3           X4
## Min.      :16.00   Min.      :1.000   Min.      :1.00   Min.      : 0.000
## 1st Qu.:26.00   1st Qu.:2.000   1st Qu.:3.00   1st Qu.: 1.000
## Median :32.00   Median :3.000   Median :4.00   Median : 3.000
## Mean     :32.54   Mean     :2.959   Mean     :3.43   Mean     : 3.261
## 3rd Qu.:39.00   3rd Qu.:4.000   3rd Qu.:4.00   3rd Qu.: 4.000
## Max.     :49.00   Max.     :4.000   Max.     :4.00   Max.     :16.000
##           X5           X6           X7           X8
## Min.      :0.0000   Min.      :0.0000   Min.      :1.000   Min.      :1.000
## 1st Qu.:1.0000   1st Qu.:0.0000   1st Qu.:1.000   1st Qu.:3.000
## Median :1.0000   Median :1.0000   Median :2.000   Median :3.000
## Mean     :0.8506   Mean     :0.7495   Mean     :2.138   Mean     :3.134
## 3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:3.000   3rd Qu.:4.000
## Max.     :1.0000   Max.     :1.0000   Max.     :4.000   Max.     :4.000
##           X9           X10
## Min.      :0.000   Min.      :1.00
## 1st Qu.:0.000   1st Qu.:1.00
## Median :0.000   Median :2.00
## Mean     :0.074   Mean     :1.92
## 3rd Qu.:0.000   3rd Qu.:3.00
## Max.     :1.000   Max.     :3.00
```

## 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