Working With R Data

Russell Almond

9/4/2020

Objectives

At the end of this lesson you should be able to

- Make vectors in R
- Access parts of the vector using the `[]` operator.
- Numeric Indexes
- Negative Indexes
- Logical Indexes
- Character Indexes
- Check types of object using `is` and `mode` functions.
- Convert types of object using `as` functions.
- Access names elements of lists using `$`.
- Access elements, row and columns of matrixes using `[,]`
- Convert between data frames and matrixes
- Read and write data frames using `read.csv` and `write.csv`.

This lesson covers the traditional R way of doing things. The next lesson will show tidyverse alternatives.

R Objects containing data

Basic R Container objects

- Vector – ordered collection of objects of the same storage `mode` (`[` extract)
- Named Vector – adds a `names` attribute (Can use names in subscripts)
- Matrix, Array – adds a `dim` and `dimnames` attribute
- List – ordered collection of objects of any type or mode (`[[` extract)
- Named List – add `names` attribute (Can use `$` to extract elements)
- S3 Class – adds a `class` attribute
- data.frame – a list of columns in a spreadsheet. Uses (`[` or `$` to extract).
- tibble – The tidyverse extension of a data frame.
- S4 Class – formal class mechanism. Uses `@` instead of `$`.

Storage modes.

The mode function in R refers to storage modes, not the mode of a distribution.

```
mode(123)
```

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

```
mode(123L)
```

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

```
mode(TRUE)
```

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

```
mode("True")
```

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

```
mode(3.14)
```

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

```
mode(t)
```

```
## [1] "function"
```

```
?mode
```

- The `is.XXX` functions can be used to check the type (mode or class) of an object.
- The `as.XXX` functions can be used to convert between different types.

```
is.integer(3)
```

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

```
is.integer(3L)
```

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

```
as.integer(3)
```

```
## [1] 3
```

```
is.integer(as.integer(3))
```

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

```
as.integer("three")
```

```
## Warning: NAs introduced by coercion
```

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

```
as.character(3)
```

```
## [1] "3"
```

```
as.logical(3)
```

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

The most commonly seen modes are:

- Numeric
- Real (the default)
- Integer (Putting an `L` after a number tells R that this should be an integer.)
- Logical (`TRUE`/`T` or `FALSE`/`F`)
- Character – Each element of a character vector is a string.
- Any – A vector of anything is a list; thus, almost all R objects are in fact vectors.

Factors

- The `factor` and `ordered` classes also behave a lot like storage modes.
- Atucally, they are R classes where the data values are integers and there is a special property which gives the names of the levels.
- The built-in data value `state.region` is a factor.

*(The function `head()` lists the first 6 data points instead of all of them.)

```
head(state.region)
```

```
## [1] South West  West  South West  West
## Levels: Northeast South North Central West
```

```
levels(state.region)
```

```
## [1] "Northeast"     "South"         "North Central" "West"
```

```
head(as.integer(state.region))
```

```
## [1] 2 4 4 2 4 4
```

```
head(as.character(state.region))
```

```
## [1] "South" "West"  "West"  "South" "West"  "West"
```

```
unclass(state.region)
```

```
##  [1] 2 4 4 2 4 4 1 2 2 2 4 4 3 3 3 3 2 2 1 2 1 3 3 2 3 4 3 4 1 1 4 1 2 3 3 2 4 1
## [39] 1 2 3 2 2 4 1 2 4 2 3 4
## attr(,"levels")
## [1] "Northeast"     "South"         "North Central" "West"
```

- The values of a factor variable are just labels,
- Numeric labels
  - `as.integer()`
- String labels
  - `as.character()`
- The function `as.factor()` will force a numeric or character vector into a factor.
- R will just pick and arbitrary order (usually alphabetical) for labels.
- Alphabetical ordering doesn't always work with `as.ordered()`.
  - `High`,`Low`,`Medium`
  - Use the function `ordered()` with more control over the levels.

```
help(ordered)
ofact <- ordered(c("H","M","H","L","M","H"),levels=c("L","M","H"))
ofact
```

```
## [1] H M H L M H
## Levels: L < M < H
```

Vectors

All R objects are vectors: scalars in R are vectors of length 1.

```
cat("The output will start with '[1]' to show that this is a vector.\n")
```

```
## The output will start with '[1]' to show that this is a vector.
```

```
3.14159
```

```
## [1] 3.14159
```

Making vectors

The `:` operator produces sequences (of integers) between first and second argument. (The function `seq()` allows step sizes of other than one.)

```
1:3
```

```
## [1] 1 2 3
```

```
3:1
```

```
## [1] 3 2 1
```

```
-1:1
```

```
## [1] -1  0  1
```

```
-3:-1
```

```
## [1] -3 -2 -1
```

The `c()` function can be used to glue vectors together. (`c` stands for combine)

```
c(1:3, 10:12)
```

```
## [1]  1  2  3 10 11 12
```

```
c("Hansel", "Gretel", "Tedd","Alice")
```

```
## [1] "Hansel" "Gretel" "Tedd"   "Alice"
```

Implicit Looping

R implicitly loops over all the elements of a vector. Such implicit loops are faster than explicit `for` loops.

```
1:11
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11
```

```
(1:11)/2
```

```
##  [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5
```

```
mean(1:11)
```

```
## [1] 6
```

```
y <- (1:11 - mean(1:11))/sd(1:11)
mean (y)
```

```
## [1] 0
```

```
sd(y)
```

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

Random vectors

- R has a number of built in random number generators to generate random numbers.
- The most commonly used are `runif`, `rnorm` and `sample`.
- Sample has a `replace` option to do sampling with or without replacement.
- There are also many others, with names that look like `rXXX` (try substituting chisq, t, beta, gamma, &c for XXX).

```
runif(5)
```

```
## [1] 0.3843185 0.5063807 0.6265046 0.8008626 0.5675777
```

```
rnorm(10)
```

```
##  [1]  1.00324952  1.30317722  0.81784953  0.81681584 -0.49723344  0.20334999
##  [7]  0.00392584 -0.70135630  0.23433767 -0.04201772
```

```
sample.int(5,5,replace=TRUE)
```

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

Exercises

1. Generate 100 random numbers with mean 50 and standard deviation 25.

1a. Use the result of the previous question to generate a random sample of size 101 with one outlier of 200.

2. Generate random integers between 0 and 100
3. The variable `state.area` contains the areas of the 50 US states (in alphabetical area). Create a random sample of size 10 of the state areas.

Three ways of subscripting a vector

- The `[]` operator is used to subscript vectors.
- There are three different things you can put inside of the brackets:
- numbers,
  - negative numbers (exclude values)
- logical expressions
- names (character values).

Numeric Indexes

- Numbers are the most straightforward way to do indexing.
- R starts the indexes at 1 and it goes up to the length of the vector.
- The function `length()` is useful in writing indexes.

- Giving multiple indexes with return a sub-vector (remember, there are no scalars in R, just vectors of length 1).

```
int10 <- 1:10
int10[3]
```

```
## [1] 3
```

```
int10[c(5:7,9)]
```

```
## [1] 5 6 7 9
```

```
state.area[c(1,length(state.area))]
```

```
## [1] 51609 97914
```

Another useful trick is to use negative indexes. These leave the numbered variables out.

```
int10[-2]
```

```
## [1]  1  3  4  5  6  7  8  9 10
```

```
int10[-(3:8)]
```

```
## [1]  1  2  9 10
```

Indexing expressions can also be used on the LHS of assignment operators, to allow to assignment to just certain values.

```
int10[3] <- -3
int10
```

```
##  [1]  1  2 -3  4  5  6  7  8  9 10
```

Logical Indexes

The second option for indexing is to use a logical vector the same length as the vector you are indexing.

```
int10<0
```

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

```
int10[int10<0]
```

```
## [1] -3
```

```
int10[int10<0] <- abs (int10[int10<0])
int10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Be careful with NAs.

```
int55 <- -5:5
sqrt(int55) < 1.2
```

```
## Warning in sqrt(int55): NaNs produced
```

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

```
int55[sqrt(int55) < 1.2]
```

```
## Warning in sqrt(int55): NaNs produced
```

```
## [1] NA NA NA NA NA  0  1
```

The real power of logical indexes comes when we have two vectors of the same length.

For example, `state.abb` gives the two letter postal codes of the states. Suppose we wanted to see all of the states that are bigger than average:

```
state.abb[state.area>median(state.area)]
```

```
##  [1] "AK" "AZ" "CA" "CO" "FL" "GA" "ID" "IL" "IA" "KS" "MI" "MN" "MO" "MT" "NE"
## [16] "NV" "NM" "ND" "OK" "OR" "SD" "TX" "UT" "WA" "WY"
```

Aside: `ifelse` and `if`

The built in language primitive `if` is **not** vectorized. It is expecting a single value. The code below will not do what you think it will.

```
if (int55 < 0) {
  cat("Negative.\n")
} else {
  cat("Non-negative.\n")
}
```

```
## Warning in if (int55 < 0) {: the condition has length > 1 and only the first
## element will be used
```

```
## Negative.
```

The functions `any()`, `all()` and `isTRUE()` are often useful here.

```
if (all(int55 >0)) {
  cat("Positive.\n")
} else {
  cat("Not all positive.\n")
}
```

```
## Not all positive.
```

The function `ifelse()` can be used to loop over if-else expressions.

- There are two differences from `if`.
- First the condition is a logical vector.
- Second, both the if-true and if-false argument are always evaluated, so they better not generate an error!

```
ifelse(int55<0, "-","+")
```

```
##  [1] "-" "-" "-" "-" "-" "+" "+" "+" "+" "+" "+"
```

Names and character indexes

It would be really convenient if we could access the state data by name.

Florida is the 9 state alphabetically, but I can't remember that.

What we can do is add names to a vector. Then we can select by name.

```
names(state.area) <- state.abb
head(state.area)
```

```
##     AL     AK     AZ     AR     CA     CO
##  51609 589757 113909  53104 158693 104247
```

```
head(names(state.area))
```

```
## [1] "AL" "AK" "AZ" "AR" "CA" "CO"
```

```
state.area["FL"]
```

```
##    FL
## 58560
```

```
state.area[c("NY","CA")]
```

```
##     NY     CA
##  49576 158693
```

Sometimes we need to make up names.

The `paste()` command is handy for that.

It is vectorized, so you can put a bunch of numbers in.

```
paste("Student",1:5,sep="_")
```

```
## [1] "Student_1" "Student_2" "Student_3" "Student_4" "Student_5"
```

Exercises

4. Write an expression that removes the outlier from the data you generated for 1b.
5. Suppose the data you generated for problem 1 was suppose to have a minimum score of 0 and a maximum score of 100. Fix, the data set so that all of the values are between 0 and 100.
6. Fix my positive/negative test, so that it has a 0 as well
7. Find all of the states that are bigger than Florida.
8. Generate a bunch of random integers between -10 and 10. Then turn all negative integers into NA.

Matrixes, Lists and Data Frames

Matrixes and Arrays

- A matrix is an object with rows and columns.
- An array can have any number of dimensions.
- But they all the entries need to be the same type (mode).
- There is a `dim()` attribute which shows the dimensions of the matrix.

```
dim(state.x77)
```

```
## [1] 50  8
```

```
head(state.x77)
```

```
##            Population Income Illiteracy Life Exp Murder HS Grad Frost   Area
## Alabama          3615   3624        2.1    69.05   15.1    41.3    20  50708
## Alaska            365   6315        1.5    69.31   11.3    66.7   152 566432
```

```
## Arizona          2212   4530        1.8    70.55    7.8    58.1    15 113417
## Arkansas         2110   3378        1.9    70.66   10.1    39.9    65  51945
## California      21198   5114        1.1    71.71   10.3    62.6    20 156361
## Colorado         2541   4884        0.7    72.06    6.8    63.9   166 103766
```

Getting and setting dims

- The `dim()` function is used to access the number of rows and columns.
- `dim()[1]` gets the number of rows
- `dim()[2]` gets the number of columns.
- For matrixes, the functions `nrow()` and `ncol()` are easier to remember.

Setting `dim()` will reshape a vector into a matrix or array.

```
nrow(state.x77)
```

```
## [1] 50
```

```
ncol(state.x77)
```

```
## [1] 8
```

```
int12 <- 1:12
dim(int12) <- c(3,4)
int12
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

`matrix()` and `array()` functions

- Setting the `dim()` attribute directly is not recommended (makes for hard to read code).
- Instead use `matrix()` or `array()`
- R stores matrixes in row major order (like FORTRAN, not like c).
  - Use `byrow=TRUE` to reverse this in `matrix` or `array`

```
matrix(1:12,3,4)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
matrix(1:12,3,4,byrow=TRUE)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
```

```
array(1:24,c(2,3,4))
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
##
## , , 3
##
##      [,1] [,2] [,3]
## [1,]   13   15   17
## [2,]   14   16   18
##
## , , 4
##
##      [,1] [,2] [,3]
## [1,]   19   21   23
## [2,]   20   22   24
```

Numeric and logical indexes

For matrixes and arrays, the `[]` operator does something a little bit different. In particular, `x[i,j]` picks out row i and column j.

Either the row or column selector could be

- A number or vector of numbers (pick those rows or columns)
- A negative number of vector of negative numbers (excluded those rows or columns)
- A logical vector of size `nrow(x)` or `ncol(x)` (select the rows/columns corresponding to true).
- A character vector (select rows or columns by name, see below).
- Left blank, in which case all rows/columns are selected.

If a single row or column is selected, then it turns into a vector.

```
state.x77[1:5,1:5]
```

```
##            Population Income Illiteracy Life Exp Murder
## Alabama          3615   3624        2.1    69.05   15.1
## Alaska            365   6315        1.5    69.31   11.3
## Arizona          2212   4530        1.8    70.55    7.8
## Arkansas         2110   3378        1.9    70.66   10.1
## California      21198   5114        1.1    71.71   10.3
```

```
state.x77[1:5,]
```

```
##            Population Income Illiteracy Life Exp Murder HS Grad Frost   Area
## Alabama          3615   3624        2.1    69.05   15.1    41.3    20  50708
## Alaska            365   6315        1.5    69.31   11.3    66.7   152 566432
## Arizona          2212   4530        1.8    70.55    7.8    58.1    15 113417
## Arkansas         2110   3378        1.9    70.66   10.1    39.9    65  51945
## California      21198   5114        1.1    71.71   10.3    62.6    20 156361
```

```
state.x77[9,]
```

```
## Population     Income Illiteracy   Life Exp     Murder    HS Grad      Frost
##    8277.00    4815.00       1.30      70.66      10.70      52.60      11.00
##       Area
##   54090.00
```

```
dim(state.x77[9,])
```

```
## NULL
```

```
head(state.x77[,3])
```

```
##    Alabama     Alaska    Arizona   Arkansas California   Colorado
##        2.1        1.5        1.8        1.9        1.1        0.7
```

```
state.x77[9,,drop=FALSE]
```

```
##         Population Income Illiteracy Life Exp Murder HS Grad Frost  Area
## Florida       8277   4815        1.3    70.66   10.7    52.6    11 54090
```

```
dim(state.x77[9,,drop=FALSE])
```

```
## [1] 1 8
```

dimnames and character indexes

To use character indexes with matrixes, we need to set the `rownames()` and `colnames()` of the matrix. We can also use the `dimnames()` (although this will produce a list).

```
rownames(state.x77)
```

```
##  [1] "Alabama"        "Alaska"         "Arizona"        "Arkansas"
##  [5] "California"     "Colorado"       "Connecticut"    "Delaware"
##  [9] "Florida"        "Georgia"        "Hawaii"         "Idaho"
## [13] "Illinois"       "Indiana"        "Iowa"           "Kansas"
## [17] "Kentucky"       "Louisiana"      "Maine"          "Maryland"
## [21] "Massachusetts"  "Michigan"       "Minnesota"      "Mississippi"
## [25] "Missouri"       "Montana"        "Nebraska"       "Nevada"
## [29] "New Hampshire"  "New Jersey"     "New Mexico"     "New York"
## [33] "North Carolina" "North Dakota"   "Ohio"           "Oklahoma"
## [37] "Oregon"         "Pennsylvania"   "Rhode Island"   "South Carolina"
## [41] "South Dakota"   "Tennessee"      "Texas"          "Utah"
## [45] "Vermont"        "Virginia"       "Washington"     "West Virginia"
## [49] "Wisconsin"      "Wyoming"
```

```
colnames(state.x77)
```

```
## [1] "Population" "Income"     "Illiteracy" "Life Exp"   "Murder"
## [6] "HS Grad"    "Frost"      "Area"
```

```
dimnames(state.x77)
```

```
## [[1]]
##  [1] "Alabama"        "Alaska"         "Arizona"        "Arkansas"
##  [5] "California"     "Colorado"       "Connecticut"    "Delaware"
##  [9] "Florida"        "Georgia"        "Hawaii"         "Idaho"
## [13] "Illinois"       "Indiana"        "Iowa"           "Kansas"
## [17] "Kentucky"       "Louisiana"      "Maine"          "Maryland"
```

```
## [21] "Massachusetts"  "Michigan"       "Minnesota"      "Mississippi"
## [25] "Missouri"       "Montana"        "Nebraska"       "Nevada"
## [29] "New Hampshire"  "New Jersey"     "New Mexico"     "New York"
## [33] "North Carolina" "North Dakota"   "Ohio"           "Oklahoma"
## [37] "Oregon"         "Pennsylvania"   "Rhode Island"   "South Carolina"
## [41] "South Dakota"   "Tennessee"      "Texas"          "Utah"
## [45] "Vermont"        "Virginia"       "Washington"     "West Virginia"
## [49] "Wisconsin"      "Wyoming"
##
## [[2]]
## [1] "Population" "Income"     "Illiteracy" "Life Exp"   "Murder"
## [6] "HS Grad"    "Frost"      "Area"
```

```
rownames(state.x77) <- state.abb
head(state.x77)
```

```
##    Population Income Illiteracy Life Exp Murder HS Grad Frost   Area
## AL       3615   3624        2.1    69.05   15.1    41.3    20  50708
## AK        365   6315        1.5    69.31   11.3    66.7   152 566432
## AZ       2212   4530        1.8    70.55    7.8    58.1    15 113417
## AR       2110   3378        1.9    70.66   10.1    39.9    65  51945
## CA      21198   5114        1.1    71.71   10.3    62.6    20 156361
## CO       2541   4884        0.7    72.06    6.8    63.9   166 103766
```

Row and column sums and averages

Remember that a matrix is just a vector with a `dim` attribute. Consequently, `mean` and other summary functions don't do what we want:

```
mean(state.x77)
```

```
## [1] 9956.887
```

```
sd(state.x77)
```

```
## [1] 37801.78
```

```
var(state.x77)
```

```
##               Population        Income   Illiteracy      Life Exp       Murder
## Population 19931683.7588   571229.7796  292.8679592 -4.078425e+02  5663.523714
## Income       571229.7796   377573.3061 -163.7020408  2.806632e+02  -521.894286
## Illiteracy      292.8680     -163.7020    0.3715306 -4.815122e-01     1.581776
## Life Exp       -407.8425      280.6632   -0.4815122  1.802020e+00    -3.869480
## Murder         5663.5237     -521.8943    1.5817755 -3.869480e+00    13.627465
## HS Grad       -3551.5096     3076.7690   -3.2354694  6.312685e+00   -14.549616
## Frost        -77081.9727     7227.6041  -21.2900000  1.828678e+01  -103.406000
## Area        8587916.9494 19049013.7510 4018.3371429 -1.229410e+04 71940.429959
##                  HS Grad        Frost          Area
## Population  -3551.509551 -77081.97265  8.587917e+06
## Income       3076.768980   7227.60408  1.904901e+07
## Illiteracy     -3.235469    -21.29000  4.018337e+03
## Life Exp        6.312685     18.28678 -1.229410e+04
## Murder        -14.549616   -103.40600  7.194043e+04
## HS Grad        65.237894    153.99216  2.298732e+05
## Frost         153.992163   2702.00857  2.627039e+05
## Area       229873.192816 262703.89306  7.280748e+09
```

```
cor(state.x77)
```

```
##             Population     Income  Illiteracy    Life Exp     Murder
## Population  1.00000000  0.2082276  0.10762237 -0.06805195  0.3436428
## Income      0.20822756  1.0000000 -0.43707519  0.34025534 -0.2300776
## Illiteracy  0.10762237 -0.4370752  1.00000000 -0.58847793  0.7029752
## Life Exp   -0.06805195  0.3402553 -0.58847793  1.00000000 -0.7808458
## Murder      0.34364275 -0.2300776  0.70297520 -0.78084575  1.0000000
## HS Grad    -0.09848975  0.6199323 -0.65718861  0.58221620 -0.4879710
## Frost      -0.33215245  0.2262822 -0.67194697  0.26206801 -0.5388834
## Area        0.02254384  0.3633154  0.07726113 -0.10733194  0.2283902
##                HS Grad      Frost        Area
## Population -0.09848975 -0.3321525  0.02254384
## Income      0.61993232  0.2262822  0.36331544
## Illiteracy -0.65718861 -0.6719470  0.07726113
## Life Exp    0.58221620  0.2620680 -0.10733194
## Murder     -0.48797102 -0.5388834  0.22839021
## HS Grad     1.00000000  0.3667797  0.33354187
## Frost       0.36677970  1.0000000  0.05922910
## Area        0.33354187  0.0592291  1.00000000
```

Taking row and column sums are such a frequent operation, that there is a shortcut for them: `rowSums()`, `colSums()`, `rowMeans()`, `colMeans()`

```
colMeans(state.x77)
```

```
## Population     Income Illiteracy   Life Exp     Murder    HS Grad      Frost
##  4246.4200  4435.8000     1.1700    70.8786     7.3780    53.1080   104.4600
##       Area
## 70735.8800
```

The `summary` function in the `tidyverse` package is another way to do this.

Exercises:

9. Find the population for all states whose area is bigger than Florida's.
10. Calculate the population density (population per area) for each state
11. Turn the state.x77 data into z-scores by subtracting the column means and dividing by the column standard deviations.
12. Scale the state.x77 data from 0 (minimum in the column) to 1 (maximum in the column).

Lists

- A list in R is a special vector whose elements can be anything, even other lists.
- It is possible to build up quite complex objects from lists (Old S3 class system.)

Use the 'list() constructor to make lists

```
list(1,2:3,"four",quote(2+3))
```

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

```
## [[2]]
## [1] 2 3
##
## [[3]]
## [1] "four"
##
## [[4]]
## 2 + 3
```

Notice that the second element is a vector and the last element is an R expression (this is what `quote` does). R lists are quite flexible.

Notice that the list is show with a double square bracket `[[` instead of a single one `[`. This is because with lists the extraction operators behave a little bit differently. The single bracket refers to a sublist, and the double bracket to the element. Fortunately, this doesn't come up a lot at the beginning because, most people use the `$` extractors instead.

Named Lists and $ extraction

Named lists have a special role in R. They are similar to environments in that they allow the analyst to associate names and values. If `x` is a list then `x[[name]]` or `x$name` will retrieve (or set if used with `<-`) that element.

```
alist <- list(one=1, two=2:3, three="three", four=quote(2+2))
alist
```

```
## $one
## [1] 1
##
## $two
## [1] 2 3
##
## $three
## [1] "three"
##
## $four
## 2 + 2
```

```
alist$two
```

```
## [1] 2 3
```

```
alist$two <- 2
alist
```

```
## $one
## [1] 1
##
## $two
## [1] 2
##
## $three
## [1] "three"
##
## $four
## 2 + 2
```

Lists and Classes

This ability to associate names and values is very hand. The older S3 (informal) class system just uses lists with appropriate values as classes. To get components, just use the `$` operator.

For example, the function `lm()` does a regression and returns an object of class `lm`. The `$` operator can be used to access its components.

```
fit1 <- lm(dist~speed,data=cars)
fit1$coefficients
```

```
## (Intercept)       speed
##  -17.579095    3.932409
```

Data frame

- A data frame is a list that behaves like a matrix.
- A data frame is a list of columns with a class of `data.frame`.
- Different columns can have different classes or storage modes.
- Matrixes and arrays all must be the same kind of value.
- Using the single square bracket `[i,j]` can reference row i and column j, like a matrix.
- Using the `$` operator can reference columns.

```
?mtcars
names(mtcars)  # Get the variable names
```

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

```
rownames(mtcars) # Get the car names
```

```
##  [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"
```

```
mtcars[1:5,] # First five rows
```

```
##                   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
```

```
mtcars["Honda Civic", ]  # Just one car
```

```
##             mpg cyl disp hp drat    wt  qsec vs am gear carb
## Honda Civic 30.4   4 75.7 52 4.93 1.615 18.52  1  1    4    2
```

```
mtcars[,"mpg"] # Just MPG variable
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
```

```
mtcars$disp  # Just DISP variable
```

```
##  [1] 160.0 160.0 108.0 258.0 360.0 225.0 360.0 146.7 140.8 167.6 167.6 275.8
## [13] 275.8 275.8 472.0 460.0 440.0  78.7  75.7  71.1 120.1 318.0 304.0 350.0
## [25] 400.0  79.0 120.3  95.1 351.0 145.0 301.0 121.0
```

data.frame(), as.matrix and as.data.frame

The function `data.frame()` will put a data frame together column by column. (If one of the arguments is a matrix each column in the matrix will become a column in the data frame.)

```
stateX77 <- data.frame(state.x77,region=state.region,row.names=state.abb)
stateX77
```

```
##    Population Income Illiteracy Life.Exp Murder HS.Grad Frost   Area
## AL       3615   3624        2.1    69.05   15.1    41.3    20  50708
## AK        365   6315        1.5    69.31   11.3    66.7   152 566432
## AZ       2212   4530        1.8    70.55    7.8    58.1    15 113417
## AR       2110   3378        1.9    70.66   10.1    39.9    65  51945
## CA      21198   5114        1.1    71.71   10.3    62.6    20 156361
## CO       2541   4884        0.7    72.06    6.8    63.9   166 103766
## CT       3100   5348        1.1    72.48    3.1    56.0   139   4862
## DE        579   4809        0.9    70.06    6.2    54.6   103   1982
## FL       8277   4815        1.3    70.66   10.7    52.6    11  54090
## GA       4931   4091        2.0    68.54   13.9    40.6    60  58073
## HI        868   4963        1.9    73.60    6.2    61.9     0   6425
## ID        813   4119        0.6    71.87    5.3    59.5   126  82677
## IL      11197   5107        0.9    70.14   10.3    52.6   127  55748
## IN       5313   4458        0.7    70.88    7.1    52.9   122  36097
## IA       2861   4628        0.5    72.56    2.3    59.0   140  55941
## KS       2280   4669        0.6    72.58    4.5    59.9   114  81787
## KY       3387   3712        1.6    70.10   10.6    38.5    95  39650
## LA       3806   3545        2.8    68.76   13.2    42.2    12  44930
## ME       1058   3694        0.7    70.39    2.7    54.7   161  30920
## MD       4122   5299        0.9    70.22    8.5    52.3   101   9891
```

```
## MA       5814   4755        1.1    71.83    3.3    58.5   103   7826
## MI       9111   4751        0.9    70.63   11.1    52.8   125  56817
## MN       3921   4675        0.6    72.96    2.3    57.6   160  79289
## MS       2341   3098        2.4    68.09   12.5    41.0    50  47296
## MO       4767   4254        0.8    70.69    9.3    48.8   108  68995
## MT        746   4347        0.6    70.56    5.0    59.2   155 145587
## NE       1544   4508        0.6    72.60    2.9    59.3   139  76483
## NV        590   5149        0.5    69.03   11.5    65.2   188 109889
## NH        812   4281        0.7    71.23    3.3    57.6   174   9027
## NJ       7333   5237        1.1    70.93    5.2    52.5   115   7521
## NM       1144   3601        2.2    70.32    9.7    55.2   120 121412
## NY      18076   4903        1.4    70.55   10.9    52.7    82  47831
## NC       5441   3875        1.8    69.21   11.1    38.5    80  48798
## ND        637   5087        0.8    72.78    1.4    50.3   186  69273
## OH      10735   4561        0.8    70.82    7.4    53.2   124  40975
## OK       2715   3983        1.1    71.42    6.4    51.6    82  68782
## OR       2284   4660        0.6    72.13    4.2    60.0    44  96184
## PA      11860   4449        1.0    70.43    6.1    50.2   126  44966
## RI        931   4558        1.3    71.90    2.4    46.4   127   1049
## SC       2816   3635        2.3    67.96   11.6    37.8    65  30225
## SD        681   4167        0.5    72.08    1.7    53.3   172  75955
## TN       4173   3821        1.7    70.11   11.0    41.8    70  41328
## TX      12237   4188        2.2    70.90   12.2    47.4    35 262134
## UT       1203   4022        0.6    72.90    4.5    67.3   137  82096
## VT        472   3907        0.6    71.64    5.5    57.1   168   9267
## VA       4981   4701        1.4    70.08    9.5    47.8    85  39780
## WA       3559   4864        0.6    71.72    4.3    63.5    32  66570
## WV       1799   3617        1.4    69.48    6.7    41.6   100  24070
## WI       4589   4468        0.7    72.48    3.0    54.5   149  54464
## WY        376   4566        0.6    70.29    6.9    62.9   173  97203
##              region
## AL            South
## AK             West
## AZ             West
## AR            South
## CA             West
## CO             West
## CT        Northeast
## DE            South
## FL            South
## GA            South
## HI             West
## ID             West
## IL North Central
## IN North Central
## IA North Central
## KS North Central
## KY            South
## LA            South
## ME        Northeast
## MD            South
## MA        Northeast
## MI North Central
## MN North Central
```

```
## MS         South
## MO North Central
## MT          West
## NE North Central
## NV          West
## NH     Northeast
## NJ     Northeast
## NM          West
## NY     Northeast
## NC         South
## ND North Central
## OH North Central
## OK         South
## OR          West
## PA     Northeast
## RI     Northeast
## SC         South
## SD North Central
## TN         South
## TX         South
## UT          West
## VT     Northeast
## VA         South
## WA          West
## WV         South
## WI North Central
## WY          West
```

```
stateX77$Income
```

```
##  [1] 3624 6315 4530 3378 5114 4884 5348 4809 4815 4091 4963 4119 5107 4458 4628
## [16] 4669 3712 3545 3694 5299 4755 4751 4675 3098 4254 4347 4508 5149 4281 5237
## [31] 3601 4903 3875 5087 4561 3983 4660 4449 4558 3635 4167 3821 4188 4022 3907
## [46] 4701 4864 3617 4468 4566
```

The functions `as.data.frame()` and `as.matrix()` can be used to go back and forth between the two different representations.

- All matrixes can be converted to data frames, but data frames can only be converted to matrixes if all of the variables are the same type.
- There are certain mathematical operators (like taking the inverse) which only work on matrixes.

For most of what I do in R, the data frame is the most convenient representation for data.

The `tidyverse` package uses the `tibble` instead of the `data.frame`. A `tibble` is a new class for data frames which has slightly more intelligence printing and more consistent subseting behavior.

read.table and read.csv

Most common format for storing data is tab separated value (`.dat`) and comma separated value (`.csv`).

- Cases are rows
- Variables are separated by tab or comma
- Often a header row giving variable names
- Sometimes there are row names.
- Sometimes quotes are used for strings

The functions `read.table()` and `read.csv()` read these data files and produce data frames. * Really the same function with different options. * Many options, look at the help!!

```
help(read.table)
```

These functions automatically convert strings to factors. The `as.is` optional argument suppresses that. Often factors, dates and missing values need to be cleaned up after reading in the data. (More about this in the next lesson).

Windows Only. Usually both `.dat` and `.csv` files are mapped to open in Excel when you double click on them. If the file is open in Excel, then Windows will lock the file and not let another program read it. You may need to close the file in Excel before you can read it into R.

The functions `write.table()` and `write.csv()` go in the opposite directions.

The `tidyverse` alternative is `read_csv()`. It might be somewhat easier to use, but it produces tibbles instead of data frames. More about this in the next session.

Foreign interfaces

R can read data from an other packages, but you need to load the `foreign` package first.

- `library(foreign)` (Part of the base R distribution)
- `read.spss` (SPSS)
- `read.dta` (Stata)
- `read.ssd` (SAS)
- `read.systat` (Systat)

Excel workbooks are another common format. The easiest way to work with Excel data is to save it in `.csv` format from Excel. You could also try the `xlsx` package (need to install it first).

- `library(xlsx)` (Need to install from CRAN)
- `read.xlsx` (Excel)

The book *R for Data Science* (Grolemund and Wickham, 2017) recommends the `haven` and `readxl` packages. Also, the `DBI` package allows importing data directly from databases (an advanced R trick).

Exercises

Use the function `write.csv()` to write out the `stateX77` data we made. Read it into Excel (or another spreadsheet) make some changes. Now read the modified version back into R.

Next Lesson

You are now read for Exploratory Data Analysis with Tidyverse and GGplot.