Data types

Daniel Anderson
Week 1, Class 2

UNIVERSITY OF
OREGON

Agenda

- Finishing up on coercion
- Attributes
- Missing values
- Intro to lists
- Subsetting

Learning objectives

- Understand the fundamental difference between lists and atomic vectors

Learning objectives

- Understand the fundamental difference between lists and atomic vectors
- Understand how atomic vectors are coerced, implicitly or explicitly

Learning objectives

- Understand the fundamental difference between lists and atomic vectors
- Understand how atomic vectors are coerced, implicitly or explicitly
- Understand various ways to subset vectors, and how subsetting differs for lists

Learning objectives

- Understand the fundamental difference between lists and atomic vectors
- Understand how atomic vectors are coerced, implicitly or explicitly
- Understand various ways to subset vectors, and how subsetting differs for lists
- Understand what an attribute is, and how to set and modify attributes

Pop quiz

Without actually running the code, talk with your neighbor: Which will each of the following coerce to?

```
c(1.25, TRUE, 4L)

c(1L, FALSE)

c(7L, 6.23, "eight")

c(TRUE, 1L, 0L, "False")
```

Answers

```
typeof(c(1.25, TRUE, 4L))
```

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

```
typeof(c(1L, FALSE))
```

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

```
typeof(c(7L, 6.23, "eight"))
```

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

```
typeof(c(TRUE, 1L, 0L, "False"))
```

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

Challenge

Work with a partner

- Create four atomic vectors, one for each of the fundamental types
- Combine two or more of the vectors. Predict the implicit coercion of each.
- Apply explicit coercions to a different type, and predict the output for each.

(basically quiz each other)

Attributes

- What are attributes?

Attributes

- What are attributes?
    - metadata... what's metadata?

Attributes

- What are attributes?
    - metadata... what's metadata?
    - Data about the data

Other data types

- Atomic vectors by themselves make up only a small fraction of the total number of data types in R

Other data types

- Atomic vectors by themselves make up only a small fraction of the total number of data types in R

What are some other data types?

Other data types

- Atomic vectors by themselves make up only a small fraction of the total number of data types in R

What are some other data types?

- Data frames (actually built from lists and atomic vectors)

Other data types

- Atomic vectors by themselves make up only a small fraction of the total number of data types in R

What are some other data types?

- Data frames (actually built from lists and atomic vectors)

- Matrices & arrays

Other data types

- Atomic vectors by themselves make up only a small fraction of the total number of data types in R

What are some other data types?

- Data frames (actually built from lists and atomic vectors)
- Matrices & arrays
- Factors

Other data types

- Atomic vectors by themselves make up only a small fraction of the total number of data types in R

What are some other data types?

- Data frames (actually built from lists and atomic vectors)
- Matrices & arrays
- Factors
- Dates

Other data types

- Atomic vectors by themselves make up only a small fraction of the total number of data types in R

What are some other data types?

- Data frames (actually built from lists and atomic vectors)

- Matrices & arrays

- Factors

- Dates

Remember, atomic vectors are the atoms of R. Many other data structures are built from atomic vectors.

- We use attributes to create other data types from atomic vectors

Attributes

Common

- Names
- Dimensions

Less common

- Arbitrary metadata

Examples

- See all attributes associated with a give object with `attributes`

```
attributes(iris)
```

```
## $names
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
##
## $class
## [1] "data.frame"
##
## $row.names
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101 102
## [103] 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
## [120] 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136
## [137] 137 138 139 140 141 142 143 144 145 146 147 148 149 150
```

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Get specific attribute

- Access just a single attribute by naming it within `attr`

```
attr(iris, "class")
```

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

```
attr(iris, "names")
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
```

Get specific attribute

- Access just a single attribute by naming it within `attr`

```r
attr(iris, "class")
```

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

```r
attr(iris, "names")
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
```

Note - this is not generally how you would pull the names attribute. Rather, you would use `names`.

Be specific

- Note in the prior slides, I'm asking for attributes on the entire data frame.
- Is that what I want?... maybe. But what the individual vectors may have attributes as well

Be specific

- Note in the prior slides, I'm asking for attributes on the entire data frame.
- Is that what I want?... maybe. But what the individual vectors may have attributes as well

```r
attributes(iris$Species)
```

```
## $levels
## [1] "setosa"     "versicolor" "virginica"
##
## $class
## [1] "factor"
```

```r
attributes(iris$Sepal.Length)
```

```
## NULL
```

Set attributes

- Just redefine them within `attr`

```
attr(iris$Species, "levels") <- c("Red", "Green", "Blue")

head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2     Red
## 2          4.9         3.0          1.4         0.2     Red
## 3          4.7         3.2          1.3         0.2     Red
## 4          4.6         3.1          1.5         0.2     Red
## 5          5.0         3.6          1.4         0.2     Red
## 6          5.4         3.9          1.7         0.4     Red
```

Note - you would generally not define levels this way, but it is a general method for modifying attributes.

Dimensions

- Let's create a matrix (please do it with me)

```
m <- matrix(1:6, ncol = 2)
m
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

- Notice how the matrix fills

Dimensions

- Let's create a matrix (please do it with me)

```
m <- matrix(1:6, ncol = 2)
m
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

- Notice how the matrix fills
- Check out the attributes

```
attributes(m)
```

```
## $dim
## [1] 3 2
```

Modify the attributes

- Let's change it to a 2 x 3 matrix, instead of 3 x 2 (you try first)

Modify the attributes

- Let's change it to a 2 x 3 matrix, instead of 3 x 2 (you try first)

```r
attr(m, "dim") <- c(2, 3)
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Modify the attributes

- Let's change it to a 2 x 3 matrix, instead of 3 x 2 (you try first)

```
attr(m, "dim") <- c(2, 3)
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

- is this the result what you expected?

Alternative creation

- Create an atomic vector, assign a dimension attribute

```
v <- 1:6
v
```

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

```
attr(v, "dim") <- c(3, 2)
v
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

Aside

- What if we wanted it to fill by row?

```r
matrix(6:13,
       ncol = 2,
       byrow = TRUE)
```

```
##      [,1] [,2]
## [1,]    6    7
## [2,]    8    9
## [3,]   10   11
## [4,]   12   13
```

```r
vect <- 6:13
dim(vect) <- c(2, 4)
vect
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    6    8   10   12
## [2,]    7    9   11   13
```

```r
t(vect)
```

```
##      [,1] [,2]
## [1,]    6    7
## [2,]    8    9
## [3,]   10   11
## [4,]   12   13
```

Names

- The following (this slide and the next) are equivalent

```
attr(v, "dimnames") <- list(c("row1", "row2", "row3"),
                            c("col1", "col2"))
v
```

```
##      col1 col2
## row1    1    4
## row2    2    5
## row3    3    6
```

Names

```r
v2 <- 1:6
attr(v2, "dim") <- c(3, 2)
rownames(v2) <- c("row1", "row2", "row3")
colnames(v2) <- c("col1", "col2")
v2
```

```
##      col1 col2
## row1    1    4
## row2    2    5
## row3    3    6
```

Arbitrary metadata

- I don't use this often (wouldn't recommend you do either)

```r
attr(v, "matrix_mean") <- mean(v)
v
```

```
##      col1 col2
## row1    1    4
## row2    2    5
## row3    3    6
## attr(,"matrix_mean")
## [1] 3.5
```

```r
attr(v, "matrix_mean")
```

```
## [1] 3.5
```

- Note that *anything* can be stored as an attribute (including matrices or data frames, etc.)

Stripping attributes

- Many operations will strip attributes (generally why it's not a good idea to store important things in them)

```
v
```

```
##      col1 col2
## row1    1    4
## row2    2    5
## row3    3    6
## attr(,"matrix_mean")
## [1] 3.5
```

```
rowSums(v)
```

```
## row1 row2 row3
##    5    7    9
```

Stripping attributes

- Many operations will strip attributes (generally why it's not a good idea to store important things in them)

```
v
```

```
##      col1 col2
## row1    1    4
## row2    2    5
## row3    3    6
## attr(,"matrix_mean")
## [1] 3.5
```

```
rowSums(v)
```

```
## row1 row2 row3
##    5    7    9
```

```
attributes(rowSums(v))
```

```
## $names
## [1] "row1" "row2" "row3"
```

- Generally `names` are maintained
- Sometimes, `dim` is maintained, sometimes not (notice it was not here)
- All else is stripped

More on names

- The `names` attribute corresponds to the individual elements within a vector

```
names(v)
```

```
## NULL
```

```
names(v) <- letters[1:6]
v
```

```
##      col1 col2
## row1    1    4
## row2    2    5
## row3    3    6
## attr(,"matrix_mean")
## [1] 3.5
## attr(,"names")
## [1] "a" "b" "c" "d" "e" "f"
```

- Perhaps more straightforward

```r
v3a <- c(a = 5, b = 7, c = 12)
v3a
```

```
##  a  b  c
##  5  7 12
```

```r
names(v3a)
```

```
## [1] "a" "b" "c"
```

```r
attributes(v3a)
```

```
## $names
## [1] "a" "b" "c"
```

Alternatives

```r
v3b <- c(5, 7, 12)
names(v3b) <- c("a", "b", "c")
v3b
```

```
##  a  b  c
##  5  7 12
```

```r
v3c <- setNames(c(5, 7, 12), c("a", "b", "c"))
v3c
```

```
##  a  b  c
##  5  7 12
```

Alternatives

```
v3b <- c(5, 7, 12)
names(v3b) <- c("a", "b", "c")
v3b
```

```
##  a  b  c
##  5  7 12
```

```
v3c <- setNames(c(5, 7, 12), c("a", "b", "c"))
v3c
```

```
##  a  b  c
##  5  7 12
```

- Note that `names` is **not** the same thing as `colnames`, but, somewhat confusingly, both work to rename the variables (columns) of a data frame.

Implementation of factors

Quickly

```r
fct <- factor(c("a", "a", "b", "c"))
typeof(fct)
```

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

```r
attributes(fct)
```

```
## $levels
## [1] "a" "b" "c"
##
## $class
## [1] "factor"
```

```r
str(fct)
```

```
##  Factor w/ 3 levels "a","b","c": 1 1 2 3
```

Implementation of dates

Quickly

```r
date <- Sys.Date()
typeof(date)
```

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

```r
attributes(date)
```

```
## $class
## [1] "Date"
```

```r
attributes(date) <- NULL
date
```

```
## [1] 17988
```

- This number represents the days passed since January 1, 1970, known as the Unix epoch.

Missing values

- Missing values breed missing values

```
NA > 5
```

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

```
NA * 7
```

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

Missing values

- Missing values breed missing values

```
NA > 5
```

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

```
NA * 7
```

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

- What about this one?

```
NA == NA
```

Missing values

- Missing values breed missing values

```
NA > 5
```

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

```
NA * 7
```

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

- What about this one?

```
NA == NA
```

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

Missing values

- Missing values breed missing values

```
NA > 5
```

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

```
NA * 7
```

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

- What about this one?

```
NA == NA
```

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

It is correct because there's no reason to presume that one missing value is or is not equal to another missing value.

When missing values don't propagate

```
NA | TRUE
```

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

```
x <- c(NA, 3, NA, 5)
any(x > 4)
```

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

How to test missingness?

- We've already seen the following doesn't work

```
x == NA
```

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

How to test missingness?

- We've already seen the following doesn't work

```
x == NA
```

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

- Instead, use `is.na`

```
is.na(x)
```

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

- When does this regularly come into play?

Lists

Lists

- Lists are vectors, but not *atomic* vectors

- Fundamental difference - each element can be a different type

```
list("a", 7L, 3.25, TRUE)
```

```
## [[1]]
## [1] "a"
##
## [[2]]
## [1] 7
##
## [[3]]
## [1] 3.25
##
## [[4]]
## [1] TRUE
```

Lists

- Technically, each element of the list is an atomic vector
- The prior example included all *scalars*, which are vectors of length 1.
- Lists do not require all elements to be the same length

```r
l <- list(c("a", "b", "c"),
     rnorm(5),
     c(7L, 2L),
     c(TRUE, TRUE, FALSE, TRUE))
l
```

```
## [[1]]
## [1] "a" "b" "c"
##
## [[2]]
## [1] -1.3740535  0.8621910  0.8572382 -0
##
## [[3]]
## [1] 7 2
##
## [[4]]
## [1]  TRUE  TRUE FALSE  TRUE
```

Check the list

```
typeof(l)
```

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

```
attributes(l)
```

```
## NULL
```

```
str(l)
```

```
## List of 4
##  $ : chr [1:3] "a" "b" "c"
##  $ : num [1:5] -1.374 0.862 0.857 -0.613 -1.457
##  $ : int [1:2] 7 2
##  $ : logi [1:4] TRUE TRUE FALSE TRUE
```

Data frames as lists

- A data frame is just a special case of a list, where all the elements are of the same length.

```r
l_df <- list(a = c("red", "blue"),
             b = rnorm(2),
             c = c(7L, 2L),
             d = c(TRUE, FALSE))
l_df
```

```
## $a
## [1] "red"  "blue"
##
## $b
## [1] 2.084453 1.534013
##
## $c
## [1] 7 2
##
## $d
## [1]  TRUE FALSE
```

```r
data.frame(l_df)
```

```
##      a        b c     d
## 1  red 2.084453 7  TRUE
## 2 blue 1.534013 2 FALSE
```

Subsetting

Pop quiz

Work with your neighbor

```r
x <- c(a = 3, b = 5, c = 7)

l <- list(x = x,
          x2 = c(x, x),
          x3 = list(vect = x,
                    squared = x^2,
                    cubed = x^3))
```

- Show three different ways to extract the first element of `x` above
- Try extracting `x` from `l`. Use `typeof` to check if you actually got the vector, and not a list.
- Show one alternative method of extracting `x` from `l`. Check your result with `typeof` again.
- Try to extract the cubed version using two different methods

Answers

Three methods of extracting `x`

- Note, there are other methods too (and of course I'm showing four here, not three)

```r
x["a"]
```

```
## a
## 3
```

```r
x[1]
```

```
## a
## 3
```

```r
x[c(TRUE, FALSE, FALSE)]
```

```
## a
## 3
```

```r
x[-c(2, 3)]
```

```
## a
## 3
```

Answers

Three methods of extracting `x`

- Note, there are other methods too (and of course I'm showing four here, not three)

```
x["a"]
```

```
## a
## 3
```

```
x[1]
```

```
## a
## 3
```

```
x[c(TRUE, FALSE, FALSE)]
```

```
## a
## 3
```

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

```
## a
## 3
```

- Why does `x["a"]` work?

Aside

Be careful with factors

```r
fct
```

```
## [1] a a b c
## Levels: a b c
```

```r
fct["b"]
```

```
## [1] <NA>
## Levels: a b c
```

```r
fct[3]
```

```
## [1] b
## Levels: a b c
```

```r
fct[factor("b")]
```

```
## [1] a
## Levels: a b c
```

Subsetting lists

Multiple methods

- Most common: `$`, `[`, and `[[`

```
l[1]
```

```
## $x
## a b c
## 3 5 7
```

```
typeof(l[1])
```

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

```
l[[1]]
```

```
## a b c
## 3 5 7
```

```
typeof(l[[1]])
```

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

```
l[[1]]["c"]
```

```
## c
## 7
```

Named list

- Because the elements of the list are named, we can use $

```
l$x2
```

```
## a b c a b c
## 3 5 7 3 5 7
```

```
l$x3
```

```
## $vect
## a b c
## 3 5 7
##
## $squared
##  a  b  c
##  9 25 49
##
## $cubed
##   a   b   c
##  27 125 343
```

Subsetting nested lists

- Multiple `$` if all named

```
l$x3$squared
```

```
##   a  b  c
##   9 25 49
```

- Note this doesn't work on named elements of an atomic vector, just the named elements of a list

```
l$x3$squared$b
```

```
## Error in l$x3$squared$b: $ operator is invalid for atomic vectors
```

Subsetting nested lists

- Multiple `$` if all named

```
l$x3$squared
```

```
##  a  b  c
##  9 25 49
```

- Note this doesn't work on named elements of an atomic vector, just the named elements of a list

```
l$x3$squared$b
```

```
## Error in l$x3$squared$b: $ operator is invalid for atomic vectors
```

But could do

```
l$x3$squared["b"]
```

```
##  b
## 25
```

Alternatives

- You can always use logical

- Indexing works too

```r
l[c(TRUE, FALSE, TRUE)]
```

```
## $x
## a b c
## 3 5 7
##
## $x3
## $x3$vect
## a b c
## 3 5 7
##
## $x3$squared
##  a  b  c
##  9 25 49
##
## $x3$cubed
##   a   b   c
##  27 125 343
```

```r
l[c(1, 3)]
```

```
## $x
## a b c
## 3 5 7
##
## $x3
## $x3$vect
## a b c
## 3 5 7
##
## $x3$squared
##  a  b  c
##  9 25 49
##
## $x3$cubed
##   a   b   c
##  27 125 343
```

Careful with your brackets

```
l[[c(TRUE, FALSE, FALSE)]]
```

```
## Error in l[[c(TRUE, FALSE, FALSE)]]: recursive indexing failed at level 2
```

- Why doesn't the above work?

Subsetting in multiple dimensions

- Generally we deal with 2d data frames
- If there are two dimensions, we separate the `[` subsetting with a comma

```
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
```

```
mtcars[3, 4]
```

```
## [1] 93
```

Empty indicators

- An empty indicator implies "all"

Empty indicators

- An empty indicator implies "all"

Select the entire fourth column

```
mtcars[ ,4]
```

```
##  [1] 110 110  93 110 175 105 245  62  95 123 123 180 180 180 205 215 230
## [18]  66  52  65  97 150 150 245 175  66  91 113 264 175 335 109
```

Empty indicators

- An empty indicator implies "all"

Select the entire fourth column

```
mtcars[ ,4]
```

```
##  [1] 110 110  93 110 175 105 245  62  95 123 123 180 180 180 205 215 230
## [18]  66  52  65  97 150 150 245 175  66  91 113 264 175 335 109
```

Select the entire 4th row

```
mtcars[4, ]
```

```
##                  mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Hornet 4 Drive 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
```

Data types returned

- By default, each of the prior will return a vector, which itself can be subset

The following are equivalent

```
mtcars[4, c("mpg", "hp")]
```

```
##                  mpg  hp
## Hornet 4 Drive 21.4 110
```

```
mtcars[4, ][c("mpg", "hp")]
```

```
##                  mpg  hp
## Hornet 4 Drive 21.4 110
```

Return a data frame

- Often, you don't want the vector returned, but rather the modified data frame.

- Specify `drop = FALSE`

```
mtcars[ ,4]
```

```
##  [1] 110 110  93 110 175 105 245  62  95 123 123 180 180 180 205 215 230
## [18]  66  52  65  97 150 150 245 175  66  91 113 264 175 335 109
```

```
mtcars[ ,4, drop = FALSE]
```

```
##                      hp
## Mazda RX4           110
## Mazda RX4 Wag       110
## Datsun 710           93
## Hornet 4 Drive      110
## Hornet Sportabout   175
## Valiant             105
## Duster 360          245
## Merc 240D            62
## Merc 230             95
```

More than two dimensions

- Depending on your applications, you may not run into this much

```
array <- 1:12
dim(array) <- c(2, 3, 2)
array
```

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

Subset array

Select just the second matrix

Subset array

Select just the second matrix

```
array[ , ,2]
```

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

Subset array

Select just the second matrix

```
array[ , ,2]
```

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

Select first column of each matrix

Subset array

Select just the second matrix

```
array[ , ,2]
```

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

Select first column of each matrix

```
array[ ,1, ]
```

```
##      [,1] [,2]
## [1,]    1    7
## [2,]    2    8
```

Summary

- Atomic vectors must all be the same type
    - implicit coercion occurs if not (and you haven't specified the coercion explicitly)
- Lists are also vectors, but not atomic vectors
    - Each element can be of a different type and length
    - Incredibly flexible, but often a little more difficult to get the hang of, particularly with subsetting

Back to lists

Why are they so useful?

- Fairly obviously, they're much more flexible

- Often returned by functions, for example, `lm`

```
m <- lm(mpg ~ hp, mtcars)
str(m)
```

```
## List of 12
##  $ coefficients : Named num [1:2] 30.0989 -0.0682
##   ..- attr(*, "names")= chr [1:2] "(Intercept)" "hp"
##  $ residuals    : Named num [1:32] -1.594 -1.594 -0.954 -1.194 0.541 ...
##   ..- attr(*, "names")= chr [1:32] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" "Ho
##  $ effects      : Named num [1:32] -113.65 -26.046 -0.556 -0.852 0.67 ...
##   ..- attr(*, "names")= chr [1:32] "(Intercept)" "hp" "" "" ...
##  $ rank         : int 2
##  $ fitted.values: Named num [1:32] 22.6 22.6 23.8 22.6 18.2 ...
##   ..- attr(*, "names")= chr [1:32] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" "Ho
##  $ assign       : int [1:2] 0 1
##  $ qr           :List of 5
##   ..$ qr   : num [1:32, 1:2] -5.657 0.177 0.177 0.177 0.177 ...
##   .. ..- attr(*, "dimnames")=List of 2
```

Probably out of time but...

Challenge

```r
set.seed(123)
m <- lm(mpg ~ hp, mtcars)

l <- list(a = list(m = matrix(1:12, ncol = 3),
                   v = 1:7),
          b = data.frame(student = 1:15,
                         score = rnorm(15, 100, 10)))
```

- From the model results:
  - Extract the qr tolerance
  - Extract the term labels

- From the list
  - Extract `m`
  - Extract the third column of `m`. Maintain the matrix structure
  - Extract the score for student 7?

Note - this is meant to be very challenging. Don't worry if you struggle.

Next time

Loops with base R

Guest lecture with Dr. Joseph Nese