1-06 Lists and Data Frames

CSI 500

Course material derived from:
An Introduction to R. Notes on R: A Programming Environment for Data Analysis and Graphics
Version 3.4.3 (2017-11-30)
https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf

Lists

- Unlike single-typed arrays, lists can have elements of any type
  - character, logical, vector, matrix, or other lists
- elements accessed name
  - e$airports[1] returns "JFK"
- elements accessed by index
  - e[[3]][1] returns "JFK"
- use variable for index
  - pos = "airports"
  - e[[pos]][1] returns "JFK"
- length returns top elements
  - doesn't count sub elements

```
# list example
> e = list(name="my first list",
+ nums=1:10,
+ airports=c('JFK','ORD','LAX'),
+ data=matrix(2,2,data=4:7))
> e
$name
[1] "my first list"

$nums
 [1]  1  2  3  4  5  6  7  8  9 10

$airports
[1] "JFK" "ORD" "LAX"

$data
     [,1] [,2]
[1,]    4    6
[2,]    5    7

> length(e)
[1] 4
```

Lists

- Formed using list() function
  - convention is to use a name for each added element
  - makes accessing list elements easier, such as e$airports[2]

```
# list example
> a = "my first list"
> b = 1:10
> c = c('JFK','ORD','LAX')
> d = matrix(nrow=2,ncol=2,data=4:7)
> e = list(name=a, nums=b, airports=c, data=d)
> e
$name
[1] "my first list"

$nums
 [1]  1  2  3  4  5  6  7  8  9 10

$airports
[1] "JFK" "ORD" "LAX"

$data
     [,1] [,2]
[1,]    4    6
[2,]    5    7
```

Concatenating Lists

- Larger lists can be formed by concatenating other lists via the c() function
- all elements are collapsed into a flat structure

```
# list example
> a = "my first list"
> b = 1:10
> list_one = list(name=a, nums=b)
> c = c('JFK','ORD','LAX')
> d = matrix(nrow=2,ncol=2,data=4:7)
> list_two = list(airports=c, data=d)
>
> e = c(list_one, list_two)
> e
$name
[1] "my first list"

$nums
 [1]  1  2  3  4  5  6  7  8  9 10

$airports
[1] "JFK" "ORD" "LAX"

$data
     [,1] [,2]
[1,]    4    6
[2,]    5    7
```

Data Frames

- The data frame is a workhorse structure used in R
  - like lists, allows multiple data types
- use data.frame( ) to create data frames
  - convention is to name data element vectors
- elements accessed using $ operator and indexing
  - med.frame$pid[1] returns 3357
  - med.frame$diag_code[2] returns 217.39

```
# data frame example
# let's make some fake data for the data frame
> pid = sample(1:10000,10)
> diag_code = trunc(abs(rnorm(10)*100000))/100
> gender = sample(1:2, 10, replace=TRUE)
>
> med.frame = data.frame(pid=pid,
+ diag_code=diag_code, gender=gender)
>
> med.frame
    pid diag_code gender
1  3557   1167.84      1
2  2042    217.39      1
3  8721     19.29      2
4  3941    291.70      2
5  2610    768.97      2
6  5751    967.14      2
7  3377   1134.33      1
8  6502     19.86      2
9  2947     60.03      1
10 7083   1241.21      2
```

attach and detach

- It's often useful to specify a particular data frame for your work
  - the attach() function places your data frame at the front of the search path for object lookup by R
  - reduces need for typing… !
  - the detach() function restores everything as it was before you attached
- Note use of factor() here for categorical data
  - table() creates a cross-tabulation

```
# data frame example
# lets attach our frame
> attach(med.frame)
The following objects are masked _by_ .GlobalEnv:

    diag_code, gender, pid

> gf = factor(gender, levels=1:2, c("M", "F"))
> table(gf)
gf
M F
4 6
> detach(med.frame)
>
```

Summary

- Lists are a very general data structure
  - may contain any data type (numeric, character, logical)
  - may contain other lists
  - by convention, each list elements has a name
- Data frames are the workhorse R data structure
  - contain columns of data for measurements
  - each entry corresponds to one observation
  - columns must contain same data type
  - data frame may have different data types in different columns
  - by convention, each data frame columns has a names

manage path

- To keep track of your search path (and any attached frames or lists or packages), use the search() function
  - shows the current search path order

```
# search path example
> search()
[1] ".GlobalEnv"        "package:stats"     "package:graphics"
>
> attach( med.frame )
The following objects are masked _by_ .GlobalEnv:

    diag_code, gender, pid

> search()
 [1] ".GlobalEnv"         "med.frame"          "package:stats"
 [4] "package:graphics"
>
> detach( med.frame )
> search()
[1] ".GlobalEnv"        "package:stats"     "package:graphics"

>
```