

Data Structures II: Lists and Data Frames



UNIVERSITY OF
SAN FRANCISCO

Abbie M Popa

BSDS 100 - Intro to Data Science with R



- Lists
- Data Frames

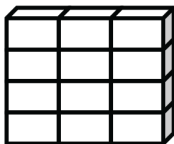
Recall: Data Structures



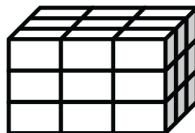
(a) Vector



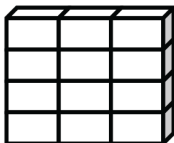
(b) Matrix



(c) Array

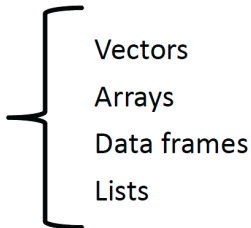


(d) Data frame



Columns can be different modes

(e) List



Part I: Lists



- Lists are different from atomic vectors as elements of a list can be of any type, including lists
- A list is constructed using `list()` instead of `c()`

```
> myList <- list(10:12, "abc", c(3.1415, 9), c(T, F, F, F))

> str(myList)
List of 4
 $ : int [1:3] 10 11 12
 $ : chr "abc"
 $ : num [1:2] 3.14 9
 $ : logi [1:4] TRUE FALSE FALSE FALSE
```



- Lists are recursive, i.e., a list can contain lists, making them fundamentally different from atomic vectors
- Handy functions

Function	Action
<code>is.list()</code>	test if list
<code>as.list()</code>	coerce to list
<code>unlist()</code>	convert to atomic vector + coercion



- Entries in a list can contain any type of data structure
- To call a single entry (say the second one) in the list `myList`, use double brackets: `myList[[2]]`
- To call multiple entries in a list (say the first and second), use single brackets: `myList[1:2]`
- If the entries in a list are named, you can call them directly using `myList$Name`

Subsetting Example



```
> myList <- list(10:12, Letters = "abc", c(3.1415, 9), Logicals =  
c(T, F, F, F))
```

```
> myList[[2]]  
[1] "abc"
```

```
> myList$Logicals  
[1] TRUE FALSE FALSE FALSE
```

```
> myList[1:2]  
[[1]]  
[1] 10 11 12
```

```
$Letters  
[1] "abc"
```


Part II: Data Frames



- Most common way of storing data in R
- A data frame is a list with equal-length vectors
- Each vector must be of the same data type

This is why we use



Data Frame Summary Example



Summary of Data `ToothGrowth`: a data frame with 60 observations on 3 variables.

- `[,1]` len numeric: Tooth length
- `[,2]` supp factor: Supplement type (VC or OJ)
- `[,3]` dose numeric: Dose in milligrams/day

```
> str(ToothGrowth)
'data.frame': 60 obs. of  3 variables:
 $ len : num  4.2 11.5 7.3 5.8 6.4 10 11.2 11.2 5.2 7 ...
 $ supp: Factor w/ 2 levels "OJ","VC": 2 2 2 2 2 2 2 2 2 2 ...
 $ dose: num  0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...

> ?ToothGrowth
```

Creating Data Frames



Create a data frame using `data.frame()`

```
# this is sloppy coding etiquette and is only for exposition
```

```
> (xyz <- data.frame(1:3, c("a", "b", "c")))
```

```
  X1.3 c..a....b....c..
```

```
1      1                a
```

```
2      2                b
```

```
3      3                c
```

```
> str(xyz)
```

```
'data.frame': 3 obs. of  2 variables:
```

```
 $ X1.3          : int  1 2 3
```

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



Create a data frame using `data.frame()`

- Surround code with `()` to automatically print the result to the console
- After creating the data frame, the first column of untitled numbers are row numbers
- Observe that even though the entries in `letterColumn` are characters that an `str(letterColumn)` shows the column to be a `Factor`



- If you want to suppress R's default behavior of turning strings into factors, use the options `stringsAsFactors = FALSE`

```
> (xyz <- data.frame(numberColumn = 1:3, letterColumn = c("a", "b", "c"),
  stringsAsFactors = F))
```

```
  numberColumn letterColumn
1             1           a
2             2           b
3             3           c
```

```
> str(xyz)
```

```
'data.frame': 3 obs. of 2 variables:
 $ numberColumn: int  1 2 3
 $ letterColumn: chr  "a" "b" "c"
```



- **Note:** A data frame is a list, which means that `typeof(myDataFrame)` will output a list
- Instead use `class()` or `is.data.frame()`
- An object can be coerced to a data frame using `as.data.frame()`



- **When a data frame already exists**, you can easily combine/append another data frame or a vector to the original data frame
- ① Use `cbind()` to column-bind two data frames
 - **Note:** the number of rows in each data frame must be equal, and row names are ignored
- ② Use `rbind()` to row-bind two data frames
 - **Note:** the **number** and the **names** of columns must match

Examples: `cbind()`



```
> (myDataFrame_01 <- data.frame(x = 1:3, y = c("A", "B", "C")))  
  x y  
1 1 A  
2 2 B  
3 3 C  
  
> (myDataFrame_02 <- cbind(myDataFrame_01, data.frame(z = -1:-3)))  
  x y z  
1 1 A -1  
2 2 B -2  
3 3 C -3
```

Examples: `rbind()`



```
> (myDataFrame_05 <- data.frame(x = 1:3, y = 98:100, z = 1000:1002))  
  x    y    z  
1 1   98 1000  
2 2   99 1001  
3 3  100 1002
```

```
> (myDataFrame_06 <- rbind(myDataFrame_05, qq = -1:-3))  
  x    y    z  
1  1   98 1000  
2  2   99 1001  
3  3  100 1002  
qq -1  -2   -3
```



- A few basic(er) `rbind()` and `cbind()` examples

Example: Try these



```
> myDataFrame_05 <- data.frame(x = 1:3, y = 98:100, z = 1000:1002)

> myDataFrame_06 <- rbind(myDataFrame_05, ???)
```

- Based on the `myDataFrame_06` code, what happens if we replace `???` with:

- (a) `qqq = -1`
- (b) `qqq = -1:-2`
- (c) `qqq = -1:-99`
- (d) `qqq = c(-1, -2)`
- (e) `qqq = c("-1", -2)`
- (f) `qqq = c("a", -2, -3)`



```
> myDataFrame_05 <- data.frame(x = 1:3, y = 98:100, z = 1000:1002)

> myDataFrame_06 <- rbind(myDataFrame_05, ???)
```

- (a) Entire additional row of -1's
- (b) Entire additional row of repeating -1's and -2's
- (c) Additional row: -1, -2, -3
- (d) Entire additional row of repeating -1's and -2's
- (e) Entire additional row of repeating -1's and -2's as **characters** (non numeric), thereby changing **all** all data frame column types to **characters**
- (f) Additional row: a, -2, -3 as **characters** (non numeric), thereby changing **all** all data frame columns types to **characters**



- Use `cbind()` to column-bind a data frame with a vector
 - **Note:** This will only work if the vector has the same length as the number of rows in the data frame.

```
> (myDataFrame_07 <- data.frame(x = 1:3, y = 98:100, z = 1000:1002))
```

	x	y	z
1	1	98	1000
2	2	99	1001
3	3	100	1002

```
> (myDataFrame_08 <- cbind(myDataFrame_05, qq = -1:-3))
```

	x	y	z	qq
1	1	98	1000	-1
2	2	99	1001	-2
3	3	100	1002	-3

Example: Try these



```
> myDataFrame_07 <- data.frame(x = 1:3, y = 98:100, z = 1000:1002)
```

```
> myDataFrame_08 <- cbind(myDataFrame_07, ???)
```

- Based on the `myDataFrame_08` code, what happens if we replace `???` with:

- (a) `qqq = -1`
- (b) `qqq = -1:-2`
- (c) `qqq = -1:-99`
- (d) `qqq = c("-1", -2)`
- (e) `qqq = c("a", -2, -3)`



```
> myDataFrame_07 <- data.frame(x = 1:3, y = 98:100, z = 1000:1002)

> myDataFrame_08 <- cbind(myDataFrame_05, ???)
```

- (a) Entire additional column of -1's
- (b) <arguments imply differing number of rows: 3, 2>
- (c) Extends the length of all other columns and repeats those values until -99
- (d) <arguments imply differing number of rows: 3, 2>
- (e) Additional column: a, -2, -3 as **factors** (non numeric)



- We will cover subsetting dataframes in more detail when we introduce tibbles!
- Most commonly, you will subset a dataframe with `$` or `[,]`
- It is also possible to use `[[]]` or `[]` (without the commas)
- Column subsetting can also be used to add new columns to the data frame