ComBEE RSG: Data Structures

M. Kartje

February 21, 2019

Review of types

R has 5 atomic data types:

Characters

```
obj<-'Thursday'
obj
```

```
## [1] "Thursday"
```

```
obj<-'This is also a character.'
obj
```

```
## [1] "This is also a character."
```

Integers and Doubles (doubles sometimes called 'numeric')

```
obj<-6
#example of a numeric
obj
```

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

```
typeof(obj)
```

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

```
obj<-as.integer(6)
#example of an integer
obj
```

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

```
typeof(obj)
```

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

Logicals

```
obj<-TRUE
#example of a logical (note lack of quotations in the definition)
obj
```

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

```
typeof(obj)
```

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

```
#OR
obj<-FALSE
obj
```

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

and complex numbers, which we won't discuss.

Data Structures in R

R is an object-oriented language:

- Data are stored in objects
- Tasks are carried out by manipulating these objects

There are lots of different ways to arrange data in R.

Each specific way is called a 'data structure.'

Vectors

The simplest type of data structure is called a `vector`.

```
v_1<-c(1, 2, 3, 4, 5)
v_1
```

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

Try making a vector of characters. Name the vector `v_2`.

Run `typeof()` on the vectors `v_1` and `v_2`.

Try making a vector of some combination of numbers and characters.

In R, lots of functions are vectorized, meaning we can just pass the function or operation vectors, instead of passing individual scalars.

```
6 + 6
```

```
## [1] 12
```

Make a third vector containing the numbers 6 through 10. Name it `v_3`.

See what happens when you add this to `v_1` using the `+` operator.

Repeat this with other operators:

- Subtraction (`-`)
- Multiplication (`*`)
- blah blah you get the point.

Indexing Vectors

The position of an entry in a vector is its *index*.

Revisiting the vector `v_3`:

```
v_3<-c(6, 7, 8, 9, 10)
```

This vector has 5 elements. We can access individual elements using `v_3[]`.

```
#print the 3rd entry.
v_3[3]
```

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

Try printing one of the entries in your character vector `v_2`.

Using `[]`, can also subset vectors.

```
#print the 2nd - 5th entries
v_3[2:5]
```

```
## [1]  7  8  9 10
```

Lists

Similarly to a `vector`, a `list` can be viewed as a sequence.

Unlike a `vector`, a `list`:

(1) can support heterogeneous data types
(2) can have >1 dimension.

```
l_1<-list(100, 54, 392, 47)
l_1
```

```
## [[1]]
## [1] 100
##
## [[2]]
## [1] 54
##
## [[3]]
## [1] 392
##
## [[4]]
## [1] 47
```

```
#print the first item of the list
l_1[1]
```

```
## [[1]]
## [1] 100
```

Make a list, `l_2`, from some combination types (*e.g.*, numbers and characters).

W.R.T. bullet (1) above, how are different types combined into a list vs. a vector?

Matricies

Matricies are simple, 2-dimensional data strucutres. R has lots of useful matrix functionality. We probably won't use them too much.

```
#a simple square matrix
m_1<-matrix(c(10, 9, 8,
              7, 6, 5,
              4, 3, 1), byrow = TRUE, nrow = 3)
m_1
```

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

The logic for subsetting matricies is similar two before, but with 2 dimensions `[ROW,COL]` .

To subset the entry in the 2nd row and 1st column, type

```
m_1[2,1]
```

```
## [1] 7
```

How might you subset an entire row or column? ##Dataframes Dataframes are a super nifty data structure

for data analysis. + 2-dimensional + Columns as variables + Rows as observations

```
#a simple dataframe
df_1<-data.frame('row_1' = c(1, 2, 3, 4, 5), 'row_2' = c(FALSE, TRUE, TRUE, FALSE, TRUE), 'row_3' = c(1
df_1
```

```
##   row_1 row_2 row_3
## 1     1 FALSE  10.7
## 2     2  TRUE  23.8
## 3     3  TRUE  98.7
## 4     4 FALSE  23.5
## 5     5  TRUE  99.7
```

They can be indexed in a manner similar to matricies.

```
df_1[2,3]
```

```
## [1] 23.8
```

Alternatively, rows can be pulled out using `$`.

```
df_1$row_2
```

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

Let's have a look at the `iris` dataframe.

```r
#look at first few rows
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
```

```r
#examine the structure
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

By subsetting with `$` and using the `typeof()` function, determine which two other data strucutres dataframes are constructed from.

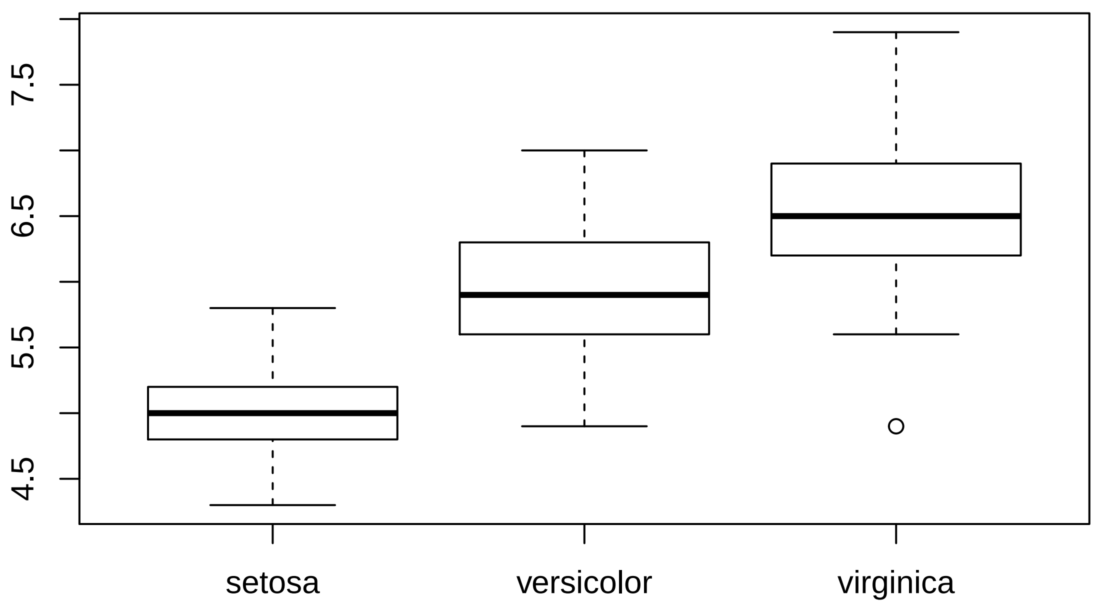
The `mean()` and `sd()` functions compute the mean and standard deviation of a vector of numbers.

Using these functions, describe the iris dataset.

Things to look forward to.

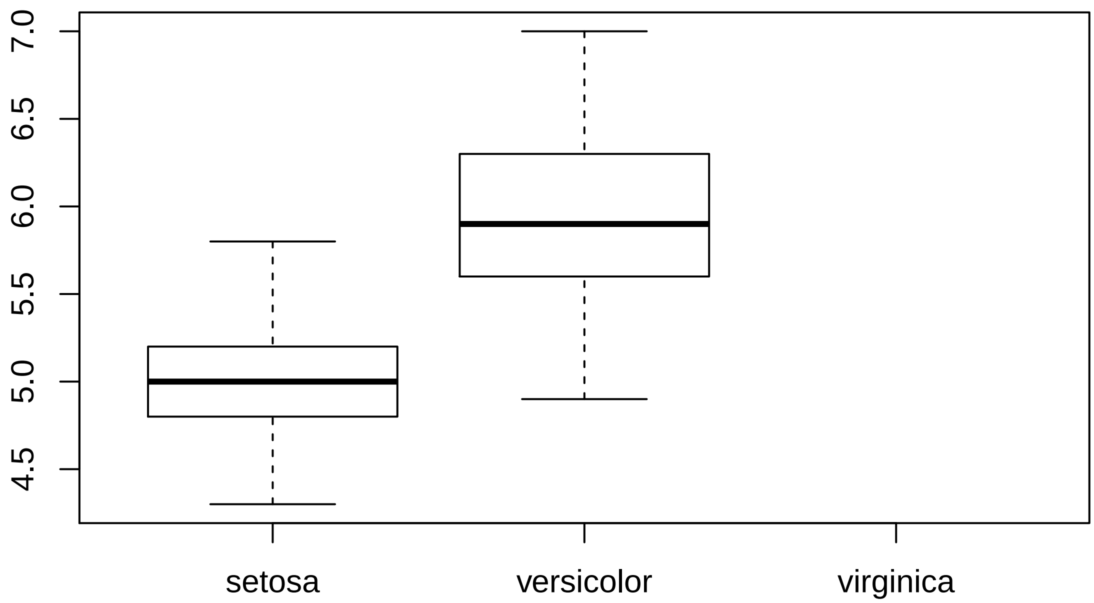
When properly organized into a dataframe, dataset can be very easy to summarize.

```r
boxplot(iris$Sepal.Length ~ iris$Species)
```



Data frames are also easy to manipulate in ways that are productive for data analysis.

```
#subset to look only at setosa and versicolor
iris_sub<-subset(iris, Species == 'setosa' | Species == 'versicolor')
boxplot(iris_sub$Sepal.Length ~ iris_sub$Species)
```




```
t.test(iris$Sepal.Length[iris$Species == 'setosa'], iris$Sepal.Length[iris$Species == 'versicolor'])
```

```
##
##  Welch Two Sample t-test
##
## data:  iris$Sepal.Length[iris$Species == "setosa"] and iris$Sepal.Length[iris$Species == "versicolor
## t = -10.521, df = 86.538, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.1057074 -0.7542926
## sample estimates:
## mean of x mean of y
##     5.006     5.936
```