Objects, Data Structures, and Data Types in R

ENVS291R v2016.11.06 Gregory S. Gilbert

Objects, Structures, and Types

- **R is object oriented**. It is organized around objects, not actions; data, not logic
- Data are bundled into **objects** on which **functions** operate
- Some functions *require* objects with specific **Data Types** or **Data Structures**
- **Data Structure** is the format into which data are arranged

  Scalar, Vector, Matrix, Data frame, List, Table, etc.

- **Data Types** are qualitative traits of the data themselves

  Integer, double, numeric, character, factor, logical

- A few tedious minutes now saves hours of head-pounding. Really.

Objects - 6 key Data Structures

There are lots of special data structures in R; 6 take you far.
1. **Scalar**: An object with a single value (a 1-element vector)
2. **Vector**: A 1D object, any length. *All elements of the same type.*
3. **Matrix**: A 2D object, in rows & columns. *All elements must be of the same type.*
4. **Data frame**: A 2D object of *mixed types*. R equivalent of a spreadsheet. Rows are records, columns are variables. Each column is of a single type, but different columns may have different types.
5. **List**: Several objects of any types strung together into a compound object. Many analytical functions produce lists.
6. **Tables**: Compact structures that store tabulated count data. Look like, but are not dataframes or matrices.

Objects - Scalars, Vectors, and Matrices

```
ascalar<-36 ; ascalar
```

```
## [1] 36
```

```
avector<-c(1,3,5,7,9) ; avector
```

```
## [1] 1 3 5 7 9
```

```
amatrix<-matrix(data=c(1,2,3,4,5,2,3,4,5,6,3,4,5,6,7),
                nrow=3,ncol=5,byrow=TRUE) ; amatrix
```

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

4/37

Objects - Scalars, Vectors, and Matrices

```
bscalar<-"A" ; bscalar
```

```
## [1] "A"
```

```
bvector<-c('A','B','C','D') ; bvector
```

```
## [1] "A" "B" "C" "D"
```

```
bmatrix<-matrix(data=c('A','B','C','D','E',
                       'B','C','D','E','F','C','D','E','F','G'),
                nrow=3,ncol=5,byrow=TRUE) ; bmatrix
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,] "A"  "B"  "C"  "D"  "E"
## [2,] "B"  "C"  "D"  "E"  "F"
## [3,] "C"  "D"  "E"  "F"  "G"
```

Getting around vectors

```
avector   #show the whole vector
```

```
## [1] 1 3 5 7 9
```

```
avector[3]  #show the 3rd element of the vector
```

```
## [1] 5
```

```
avector[2:4]  #show elements 2 through 4 of vector
```

```
## [1] 3 5 7
```

```
avector[c(1,2,5)] #show elements 1,2, and 5 of vector
```

```
## [1] 1 3 9
```

Getting around matrices

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

```
amatrix[2,]  #show the 2nd row of the matrix [row,col]
```

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

```
amatrix[,3]  #show the 3rd column of the matrix [row,col]
```

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

```
amatrix[3,c(2,4,5)] #show row 3, columns 2,4,&5 of matrix [row,col]
```

```
## [1] 4 6 7
```

Objects - A more interesting matrix

```
am<-matrix(c(63,5,84,42,2,82,51,47,8,1,24,31,6,23,71),
ncol=3,nrow=5,dimnames=list(c("elias","maria","chris","pilar","celia"
c("exam","paper","homework"))); am #create matrix of scores
```

```
##       exam paper homework
## elias   63    82       24
## maria    5    51       31
## chris   84    47        6
## pilar   42     8       23
## celia    2     1       71
```

```
str(am)  #look at the structure of the matrix
```

```
##  num [1:5, 1:3] 63 5 84 42 2 82 51 47 8 1 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:5] "elias" "maria" "chris" "pilar" ...
##   ..$ : chr [1:3] "exam" "paper" "homework"
```

Objects - str() function is your friend

```
str(am)  #look at the structure of the matrix
```

```
##  num [1:5, 1:3] 63 5 84 42 2 82 51 47 8 1 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:5] "elias" "maria" "chris" "pilar" ...
##   ..$ : chr [1:3] "exam" "paper" "homework"
```

- The matrix has [5 rows,3 columns] of numeric data
- Attribute called "dimnames" is a *list* of 2 vectors
  - 5 names of the rows
  - 3 names of the columns

```
dimnames(am)[1]  #look at the first vector of dimnames attribute
```

```
## [[1]]
## [1] "elias" "maria" "chris" "pilar" "celia"
```

Navigate a more interesting matrix

What do you get when you try these commands?

am[1,]

am["elias",]

am[,c("exam","paper")]

How would you show the exam and homework scores for the first three students? (find 4 ways)

```
##       exam homework
## elias   63       24
## maria    5       31
## chris   84        6
```

What is Celia's score on the paper?

10/37

Objects - MATRIX vs Data frame

What if we want another column for gender of student?

```
gender<-c('m','f','m','f','f') #a vector of gender
am2<-cbind(am,gender); am2 #append new column gender
```

```
##       exam paper homework gender
## elias "63" "82"  "24"     "m"
## maria "5"  "51"  "31"     "f"
## chris "84" "47"  "6"      "m"
## pilar "42" "8"   "23"     "f"
## celia "2"  "1"   "71"     "f"
```

Why the quotes?

Objects - MATRIX vs Data frame

str() is our friend

```
str(am2)
```

```
##  chr [1:5, 1:4] "63" "5" "84" "42" "2" "82" "51" "47" ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:5] "elias" "maria" "chris" "pilar" ...
##   ..$ : chr [1:4] "exam" "paper" "homework" "gender"
```

- We have a [5,4] matrix of Data Type character
- Matrix must have ALL ONE DATA TYPE
- Numbers can be character, but characters can't be numeric
- Why wasn't this a problem before with the student names?

A brief aside to Data Types

The most common *Data Types*

- *integer*: whole numbers a<-c(1,2,3,4,5)
- *double*: real numbers a<-c(3.4, -3.5, 6.3, 9.0)
- *numeric*: includes both real numbers and integers
- *character*: letters/numbers read as characters a<-c("fred","3","blue") or a<-c('fred','3', 'blue')
- *factor*[1]: a category, letters/numbers as groupings a<-c("control","N","NP","P") or a<-c('control','N','NP','P')
- *logical*: TRUE or FALSE
- *date*: letters/numbers that represent a date or time

[1] *R interprets character strings as factors by default*

Data Types: NUMERIC vs character

Let's create an object a as type **numeric**

```
a<-36 #set a equal to the numeric value 36
a # look at a
```

```
## [1] 36
```

```
str(a) #check the type of a
```

```
##  num 36
```

```
a*3 #do something with a
```

```
## [1] 108
```

Data Types: numeric vs CHARACTER

Now make object a as type character

```
a<-as.character(36) #set a equal to the numeric value 36
a # look at a
```

```
## [1] "36"
```

```
str(a) #check the type of a
```

```
##  chr "36"
```

```
a*3  #do something with it
```

```
## Error in a * 3: non-numeric argument to binary operator
```

Data Types: logical

```
a<-36  #set a to the value of 36
a==36  #does a equal 36?
```

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

```
b<-a==36 #set b to the answer of does a equal 36
b
```

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

```
str(b)
```

```
##  logi TRUE
```

Objects - MATRIX vs Data frame

- How do we add a factor column for gender of student?
- Matrices have only one data type - ojo!

```
gender<-c('m','f','m','f','f') #a vector of gender
am2<-cbind(am,gender); am2 #append new column gender
```

```
##       exam paper homework gender
## elias "63" "82"  "24"     "m"
## maria "5"  "51"  "31"     "f"
## chris "84" "47"  "6"      "m"
## pilar "42" "8"   "23"     "f"
## celia "2"  "1"   "71"     "f"
```

Objects - matrix vs DATA FRAME

```
adf<-as.data.frame(am) #convert matrix am to a data frame
gender<-c('m','f','m','f','f') #a vector of gender
adf$gender<-gender #add a new column to data frame adf
```

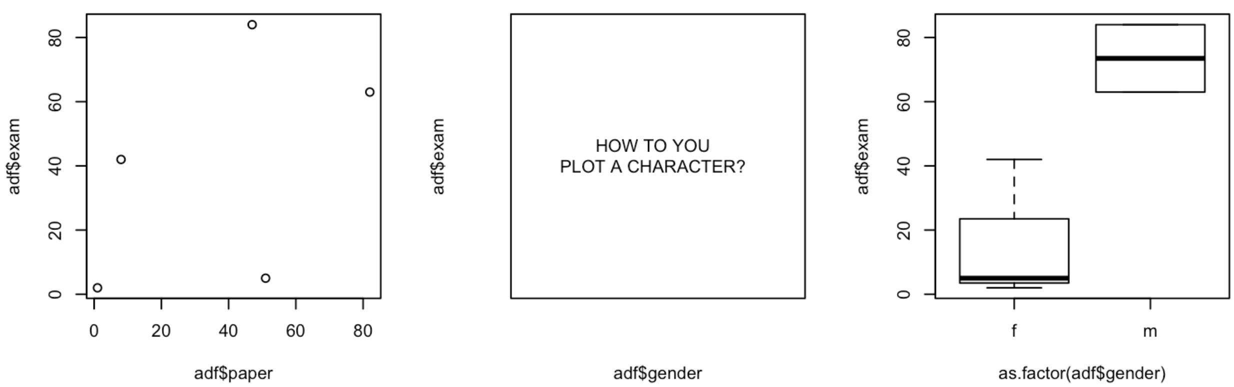
```
##       exam paper homework gender
## elias   63    82       24      m
## maria    5    51       31      f
## chris   84    47        6      m
## pilar   42     8       23      f
## celia    2     1       71      f
```

```
## 'data.frame':    5 obs. of  4 variables:
##  $ exam    : num  63 5 84 42 2
##  $ paper   : num  82 51 47 8 1
##  $ homework: num  24 31 6 23 71
##  $ gender  : chr  "m" "f" "m" "f" ...
```

DATA FRAME: character vs factor

- when you import character data, R assumes they are factors
- if you construct a dataframe as here, R treats as character
- so what?

```
plot(adf$exam~adf$paper) # plot of num vs num
plot(adf$exam~adf$gender) # plot of num vs char
plot(adf$exam~as.factor(adf$gender)) #plot of num vs factor
```

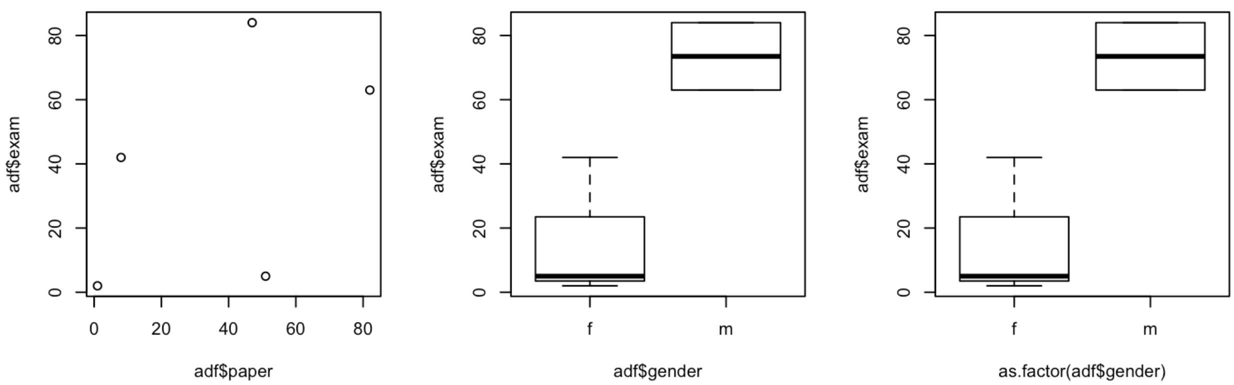


19/37

Data Types: character vs factor

```
adf$gender<-as.factor(adf$gender) #change gender to a factor
str(adf)
```

```
## 'data.frame':    5 obs. of  4 variables:
##  $ exam    : num  63 5 84 42 2
##  $ paper   : num  82 51 47 8 1
##  $ homework: num  24 31 6 23 71
##  $ gender  : Factor w/ 2 levels "f","m": 2 1 2 1 1
```



Getting around Data Frames

```
adf$paper #look at a whole column using dataframe$columnname
```

```
## [1] 82 51 47  8  1
```

```
adf$paper[3:5] #get certain records (rows) of a column
```

```
## [1] 47  8  1
```

```
adf[1:2,3:4] #can use [rows,cols] notation as for matrix
```

```
##       homework gender
## elias       24      m
## maria       31      f
```

Getting around Data Frames

```
adf$combo<-adf$exam+adf$paper #do something with columns
head(adf,3) #peek at the first 3 rows
```

```
##       exam paper homework gender combo
## elias   63    82       24      m   145
## maria    5    51       31      f    56
## chris   84    47        6      m   131
```

```
adf[adf$gender=='m',1:3] #extract based on values
```

```
##       exam paper homework
## elias   63    82       24
## chris   84    47        6
```

Objects - What is *object oriented*?

Multiply each scalar, vector, and matrix object by three

```
ascalar*3  #ascalar<-36
```

```
## [1] 108
```

```
avector*3  #avector<-c(1,3,5,7,9)
```

```
## [1]  3  9 15 21 27
```

```
amatrix*3  #amatrix<-matrix(c(1,2,3,4,5, 2,3,4,5,6, 3,...
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    3    6    9   12   15
## [2,]    6    9   12   15   18
## [3,]    9   12   15   18   21
```

Objects - What is *object oriented*?

```
adf*3 #multiply a Data Frame by a scalar
```

```
## Warning in Ops.factor(left, right): '*' not meaningful for factors
```

```
##       exam paper homework gender combo
## elias  189   246       72     NA   435
## maria   15   153       93     NA   168
## chris  252   141       18     NA   393
## pilar  126    24       69     NA   150
## celia    6     3      213     NA     9
```

The multiplier function behaves correctly depending on the Data Type of the column

Objects - What is *object oriented*?

Get the sum of each

```
sum(ascalar) #adds the one element of this scalar
```

```
## [1] 36
```

```
sum(avector) #adds the 5 elements of the vector
```

```
## [1] 25
```

```
sum(amatrix) #adds the 15 elements of the matrix
```

```
## [1] 60
```

Objects - What is *object oriented*?

Multiply each by itself

```
ascalar*ascalar  #ascalar<-36
```

```
## [1] 1296
```

```
avector*avector  #avector<-c(1,3,5,7,9)
```

```
## [1]  1  9 25 49 81
```

```
amatrix*amatrix  #amatrix<-matrix(c(1,2,3,4,5,2,3,4,5,6,3,4,5,6,7)...
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    4    9   16   25
## [2,]    4    9   16   25   36
## [3,]    9   16   25   36   49
```

Objects - What is *object oriented*?

Matrix multiplication - what do these do? How is %*% different from *?

```
avector%*%avector  #[1,3,5,7,9]•[1,3,5,7,9]
amatrix%*%avector  #amatrix•[1,3,5,7,9]
amatrix%*%t(amatrix) #amatrix•transpose(amatrix)
```

Objects - What is *object oriented*?

R is smart about applying **functions** to **objects**

```
summary(adf$exam) #summary function on numeric data
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     2.0     5.0    42.0    39.2    63.0    84.0
```

```
summary(adf$gender) #summary function on factor data
```

```
## f m
## 3 2
```

The same function does different and appropriate things depending on the structure and function of its object
> What does summary(adf) do?

Mixted type structures – Lists

Homogeneous structures

- contain elements of only one type (e.g., all numeric or all characdter).
- **Vectors** (1-dimensional) and **matrices** (2-D) are homogeneous.

Mixed-type structures

- Combine different Data Types into one structure.
- Data frame, can have different Data Types in each column.
- **Lists** string together objects of different structures and types.
- Analytical output of functions is usually a list.

Lists by example

Let's do a quick regression of exam score on paper score and put the results into the object *lmout*

```
lmout<-lm(exam~paper,data=adf); lmout
```

```
##
## Call:
## lm(formula = exam ~ paper, data = adf)
##
## Coefficients:
## (Intercept)        paper
##     19.6992       0.5159
```

This doesn't give you much, only the coefficients The real output of the regression is packed into a list

```
str(lmout) #str() is our friend
```

list - str(output of linear regression)

```
## List of 12
##  $ coefficients : Named num [1:2] 19.699 0.516
##   ..- attr(*, "names")= chr [1:2] "(Intercept)" "paper"
##  $ residuals    : Named num [1:5] 0.998 -41.01 40.054 18.174 -18.2
##   ..- attr(*, "names")= chr [1:5] "elias" "maria" "chris" "pilar"
##  $ effects      : Named num [1:5] -87.7 34.4 42.5 44.6 12.5
##   ..- attr(*, "names")= chr [1:5] "(Intercept)" "paper" "" "" ...
##  $ rank         : int 2
##  $ fitted.values: Named num [1:5] 62 46 43.9 23.8 20.2
##   ..- attr(*, "names")= chr [1:5] "elias" "maria" "chris" "pilar"
##  $ assign       : int [1:2] 0 1
##  $ qr           :List of 5
##   ..$ qr   : num [1:5, 1:2] -2.236 0.447 0.447 0.447 0.447 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:5] "elias" "maria" "chris" "pilar" ...
##   .. .. ..$ : chr [1:2] "(Intercept)" "paper"
##   .. ..- attr(*, "assign")= int [1:2] 0 1
##   ..$ qraux: num [1:2] 1.45 1.01
##   ..$ pivot: int [1:2] 1 2
##   ..$ tol  : num 1e-07
##   ..$ rank : int 2
##   ..- attr(*, "class")= chr "qr"
##  $ df.residual  : int 3
##  $ xlevels      : Named list()
##  $ call         : language lm(formula = exam ~ paper, data = adf)
##  $ terms        :Classes 'terms', 'formula'  language exam ~ paper
##   .. ..- attr(*, "variables")= language list(exam, paper)
##   .. ..- attr(*, "factors")= int [1:2, 1] 0 1
##   .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. ..$ : chr [1:2] "exam" "paper"
##   .. .. .. ..$ : chr "paper"
##   .. ..- attr(*, "term.labels")= chr "paper"
##   .. ..- attr(*, "order")= int 1
##   .. ..- attr(*, "intercept")= int 1
```

31/37

Naviating a list

```
lmout$residuals #get the named vector of residuals
lmout$coefficients[1] # get just the intercept value
lmout$coefficients["paper"] #get the slope associate with paper
lmout[['coefficients']] #use double brackets
lmout[['qr']]['qraux'] #double and single brackets to drill down
```

```
## (Intercept)       paper
##   19.699246    0.515893
```

```
## (Intercept)       paper
##   19.699246    0.515893
```

```
##    paper
## 0.515893
```

```
## $qraux
## [1] 1.447214 1.006870
```

Make a list

```
mylmout<-list(adf,lmout$coefficients[1],lmout$coefficients[2])
mylmout #look at your list of data frame, intercept, slope
```

```
## [[1]]
##       exam paper homework gender combo
## elias   63    82       24      m   145
## maria    5    51       31      f    56
## chris   84    47        6      m   131
## pilar   42     8       23      f    50
## celia    2     1       71      f     3
##
## [[2]]
## (Intercept)
##    19.69925
##
## [[3]]
##    paper
## 0.515893
```

Tables

Tables look like Data Frames or Matrices but they are not

```
table(adf$gender)
```

```
##
## f m
## 3 2
```

```
str(mytable<-table(adf$gender))
```

```
##  'table' int [1:2(1d)] 3 2
##  - attr(*, "dimnames")=List of 1
##   ..$ : chr [1:2] "f" "m"
```

Tables - a little more complicated

```
adf$hair<-as.factor(c('blonde','brown','blonde','brown','brown'))
mytable2<-table(adf$gender,adf$hair); mytable2
```

```
##
##     blonde brown
##   f      0     3
##   m      2     0
```

```
str(mytable2)
```

```
##  'table' int [1:2, 1:2] 0 2 3 0
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:2] "f" "m"
##   ..$ : chr [1:2] "blonde" "brown"
```

You cannot call the columns by their names as a data frame, but you can call out elements like you would for a matrix

35/37

Tables - a little more complicated

```
mytable2$brown  #does not work
mytable2[,2] #does work
mytable2["f",] #does work
sum(mytable2) #does work
```

But tables don't always behave like a matrix either

```
mytable2df<-as.data.frame(mytable2) #read mytable2 into a df
```

More on tables later…

Review: Objects, Structures, and Types

- R packages data into **objects**; **functions** act on objects
- The **Structure** of an object determines how to interact with it
- The **Data Type** affects what a function will do to an object
- str() is a function that shows you what an object is made of
- str() is our friend