Introduction to R and RStudio

Session II

Henry Athiany

2022-04-12

Data Objects in R

Let's learn more about objects in R

- We begin with a look at different kinds of data
  - **Booleans**: Direct binary values: `TRUE or FALSE` in R.
  - **Integers**: Whole numbers or number that can be written without fractional component, represented by a fixed-length block of bits.
  - **Characters**: fixed length block of bits with special coding.
  - **Strings**: Sequence of characters.
  - **Floating Point Numbers**: a fraction times an exponent, like $1.34x10^7$, however in R you would see `1.34e7`.
  - **Missing**: Na, NaN,

Data types

- Doubles
- Integers
- Logical
- Character
- Factor: represents categorical data. Individual code of the factor is called **level** of the factor

```
sch_educ <- factor(c("primary", "basic", "secondary"))
sch_educ
```

```
## [1] primary   basic     secondary
## Levels: basic primary secondary
```

```
levels(sch_educ)
```

```
## [1] "basic"     "primary"   "secondary"
```

Finding type of data

With types of data, R, has a built in way to help one determine the type that a certain piece of data is stored as. These consist of the following functions:

- `typeof()` this function returns the type
- `is.typ()` functions return Booleans for whether the argument is of the type `typ`
- `as.typ()` functions try to change the argument to type `typ`

```
typeof(sch_educ)
```

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

Examples

```
typeof(3)
```

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

```
is.numeric(3)
```

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

```
is.na(3)
```

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

```
is.na(3/0)
```

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

```
is.na(0/0)
```

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

```
3/0
```

```
## [1] Inf
```

Scalars

Scalars in R

- The simplest object type in R is a scalar
- A scalar object is just a single value like a number or a name

```
scalar1 <- "this is a scalar"
scalar2 <- 104
scalar3 <- 5 + 6.5  # evaluates to the single value 11.5
scalar4 <- "4"
typeof(scalar4)  # returns: character
```

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

```
## what is this type?
scalar5 <- TRUE
typeof(scalar5)  # returns: logical
```

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

Vectors

Vectors in R

What is a vector?

- The fundamental data type in R is the vector.
- It is a collection of one or more objects of the same type . We use `c()` or `vector()`
- A vector is a sequence of data elements of the same type.

Creating vectors

- What we have used here is the **concatenation** operator which takes the arguments and places them in a vector in the order in which they were entered

```
x <- c(1, 5, 2, 6)
x
```

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

```
is.vector(x)
```

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

Vectors in R

A vector can only contain objects of the same class

```
a <- c(1, 3, 4.1, 7, -1, 15)  # numeric vector
a
```

```
## [1]  1.0  3.0  4.1  7.0 -1.0 15.0
```

```
b <- c("one", "two", "three", "4")  # character vector
b
```

```
## [1] "one"   "two"   "three" "4"
```

```
c <- c("one", "two", "three", 4)  # same as above?
c
```

```
## [1] "one"   "two"   "three" "4"
```

```
d <- c(FALSE, TRUE, TRUE, FALSE, TRUE, FALSE)  #logical ve
d
```

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

Vector arithmetic

- We can do arithmetic with vectors in a similar manner as we do with integers.
- When we use operators we are doing something element by element or **elementwise**.

```
# Lets retrieve x first and then add to the new vector y
x
```

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

```
y <- c(1, 6, 4, 8)
x + y
```

```
## [1]  2 11  6 14
```

```
2 * y
```

```
## [1]  2 12  8 16
```

Elementwise

It is important to remember what happens when we consider an `elementwise` operation

```
x * y
```

```
## [1]  1 30  8 48
```

```
x/y
```

```
## [1] 1.0000000 0.8333333 0.5000000 0.7500000
```

```
x
```

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

```
y
```

```
## [1] 1 6 4 8
```

```
x%%y  #what happens here? remainder? divisor>dividend?
```

```
## [1] 0 5 2 6
```

Recycling

- We do have to be careful when performing arithmetic operations on vectors.
- There is a concept called **recycling** and this happens when 2 vectors do not have the same length

Example

```
z <- c(1, 2, 6, 8, 9, 10)
length(x)
```

```
## [1] 4
```

```
length(z)
```

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

```
# x: 1 5 2 6
x + z
```

```
## Warning in x + z: longer object length is not a multiple
## length
## [1]  2  7  8 14 10 15
```

Recycling

- Intuition would make us think that we could not perform this operation when the length of both vectors are not the same.
- However what R does is it `rewrites x` such that we have `x <- c(1 , 5, 2, 6 , 1, 5)`.
- This is called recycling, when R makes the shorter vector longer by repeating elements in the order they are listed in.

```
c(1, 5, 2, 6, 1, 5) + z
```

```
## [1]  2  7  8 14 10 15
```

Functions on vectors

- There are various functions that we can run over a vector and as we continue on we will learn more about these functions.
- One of the simplest functions can help us with knowing information about Recycling that we encountered before. This is the `length()` function
- Then length vector is very important with the writing of functions which we will get to in a later unit.
- We can use `any()` and `all()` in order to answer logical questions about elements

```
any(x > 3)
```

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

```
all(x > 3)
```

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

Built in functions

- There are various other functions that can be run on vectors, and some you are already familiar with them.
- `mean()` finds the arithmetic mean of a vector.
- `median()` finds the median of a vector.
- `sd()` and `var()` finds the standard deviation and variance of a vector respectively.
- `min()` and `max()` finds the minimum and maximum of a vector respectively.
- `sort()` returns a vector that is sorted.
- `summary()` returns a 5 number summary of the numbers in a vector.

```
# For example, finding the mean of vector x
mean(x)
```

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

which() function

- Some functions help us work with the data more to return values in which we are interested in.
- For example, above we asked if any elements in vector x were greater than 3.
- The `which()` function will tell us the elements that are.

```
which(x > 3)
```

```
## [1] 2 4
```

Vector indexing

- We can call specific elements of a vector by using the following:
  - `x[]` is a way to call up a specific element of a vector.
  - `x[1]` is the first element.
  - `x[3]` is the third element.
  - `x[-3]` is a vector with everything but the third element.

Working with vectors

```
# List elements to make sure we have what we need
x[3]
```

```
## [1] 2
```

```
x[-3]
```

```
## [1] 1 5 6
```

Replacing values

- We have seen how to subtract an element from a vector but we can use the same information to place it back in.
- We start with the same vector x that we started with.

```
x
```

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

```
x <- x[-3]
x
```

```
## [1] 1 5 6
```

Working with vectors

Inserting values

We can then add the original element back in

```
x <- c(x[1:2], 2, x[3])
x
```

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

Indexing with Booleans

- Before we used `any(x > 3)` and `which(x > 3)`.
- Now we can see not only their position in the vector, but indexing allows us to return their values.

```
x[x > 3]
```

```
## [1] 5 6
```

Naming vector elements

- With vectors it can be important to assign names to the values.
- Then when doing plots or considering maximum and minimums, instead of being given a numerical place within the vector we can be given a specific name of what that value represents.
- For example say that vector x represents the number of medications of 4 unique patients.
- We could then use the `name()` function to assign names to the values

```
x
```

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

```
names(x)
```

```
## NULL
```

```
names(x) <- c("Patient A", "Patient B", "Patient C", "Patie
x
```

```
## Patient A Patient B Patient C Patient D
```

Exercise: vectors

Exercises

1. Using a well commented R script, create a vector of the numbers `1,3,5,2,4,11,15,13,21` using the function `assign()` and then use it to perform the following tasks;

a. Find the length of the vector created
b. What type of data is this vector?
c. Sort the vector in both ascending and descending order
d. Select the third, fourth and last value of the vector

- `Hint: assign("vec",c(6,10))`

2. Using a built-in fuction `rep()`, replicate the numbers 1 to 6 two times in a vector named `test2`.

- `Hint:y <-rep(c(1:2),each = 2) or z<-rep(1:2,times = 3)`

3. The vector named `test2` is currently an `integer` type, coerce the data to character type using the function `as.character()`.

Lists

What is a List?

- Within R a list is a structure that can combine objects of different types.
- We will learn how to create and work with lists in this section.

Creating Lists

- A list is actually a vector but it does differ in comparison to the other types of vectors which we have been using in this course.
  - Other vectors are *atomic vectors*
  - A list is a type of vector called a *recursive vector*.

An Example database

- We first consider a patient database where we want to store their
  - Name
  - Amount of bill due
  - A Boolean indicator of whether or not they have insurance.

Types of information

We then have 3 types of information here:

- character
- numerical
- logical

Single patient

To create a list of one patient we say

```
a <- list(name = "Angela", owed = "75", insurance = TRUE)
a
```

```
## $name
## [1] "Angela"
##
## $owed
## [1] "75"
##
## $insurance
## [1] TRUE
```

Indexing

- Notice that unlike a typical vector this prints out in multiple parts.
- This also allows us to help with indexing as we will see later.

Lists of lists

- We could then create a list for all of our patients.
- Our database would then be a list of all of these individual lists.

List operations

- With vectors, arrays and matrices, there was really only one way to index them.
- However with lists there are multiple ways:

List indexing

```
a[["name"]]
```

```
## [1] "Angela"
```

```
a[[1]]
```

```
## [1] "Angela"
```

```
a$name
```

```
## [1] "Angela"
```

```
# double vs single brackets
a[1]  #single bracket
```

```
## $name
## [1] "Angela"
```

```
class(a[1])  #single bracket
```

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

List indexing

```
a[[1]]  #double bracket
```

```
## [1] "Angela"
```

```
class(a[[1]])  #double bracket
```

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

- With the single bracket we are returned another list.
- With the double bracket we are returned an element in the original class of what kind of data we entered.
- Depending on your goals you may want to use single or double brackets.

Adding and Subtracting elements

- With a list we can always add more information to it.

```r
a$age <- 31
a
```

```
## $name
## [1] "Angela"
##
## $owed
## [1] "75"
##
## $insurance
## [1] TRUE
##
## $age
## [1] 31
```

Adding and subtracting elements

- In order to delete an element from a list we set it to NULL.

```
a$owed <- NULL
a
```

```
## $name
## [1] "Angela"
##
## $insurance
## [1] TRUE
##
## $age
## [1] 31
```

List components and values

- To know what kind of information is included in a list, type

```
names(a)
```

```
## [1] "name"      "insurance" "age"
```

Unlisting

- To find the values of things we could go ahead and unlist them

```
a.un <- unlist(a)
a.un
```

```
##      name insurance       age
##  "Angela"    "TRUE"      "31"
```

```
class(a.un)
```

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

- If there is `Character` data in the original list that unlisted everything will be in character format.
- If your list contained all numerical elements then the class would be numerical.

Matrices

- This is a a **two-dimentional**, homogenous data structure in R. i.e. it has rows and columns
- A matrix can store data of single basic type (numeric, logical, character e.t.c)
- Thus, it can be a combination of two or more vectors
- We can create a matrix using the function `matrix()` and the syntax is

$$matrix(data, byrow, nrow, ncol, dimnames)$$

- **data**: data contains the elements in the R matrix
- **nrow/ncol**: number of rows or columns
- **byrow** : logical variable. Matrices are by default column-wise, byrow=TRUE, it does row-wise
- **dimnames**: takes two character arrays as input for row names and column names

Using matrix() function

```
# Example
mat1.data <- c(1:9)
mat1 <- matrix(mat1.data, nrow = 3, ncol = 3, byrow = TRUE)
mat1
```

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

```
# Not neccessary to always specify nrow and ncol, only one
mat2.data <- c(10:18)
mat2 <- matrix(mat2.data, nrow = 3)
mat2
```

```
##      [,1] [,2] [,3]
## [1,]   10   13   16
## [2,]   11   14   17
## [3,]   12   15   18
```

Using rbind() or cbind() function

- We can create mtrix in R by combining rows or columns as follows:

```r
mat3.data1 <- c(1, 2, 3)
mat3.data2 <- c(4, 5, 6)
mat3.data3 <- c(7, 8, 9)
mat3 <- cbind(mat3.data1, mat3.data2, mat3.data3)
mat3
```

```
##      mat3.data1 mat3.data2 mat3.data3
## [1,]          1          4          7
## [2,]          2          5          8
## [3,]          3          6          9
```

- Ex: Create a matrix using the `dim()` function. Try this;

```r
# mat4<-c(1:12); dim(mat4)<-c(4,3); mat4
```

Dataframes

Dataframes in R

- In most cases, we are likely to use the data structure called a **dataframe**.
- It is the most commonly used data structure in R
- This is similar to a matrix in appearance however we can have multiple types of data in it like a list.
- Used to store tabular data
- More general than a matrix, has different columns and can have different modes (numeric, character, factor, etc.)
- The first row is represented by **header**.
- The header is given by the list component name. Each column can store the **different datatype** which is called a **variable** and each row is an observation across multiple variables
- Each column must contain the same type of data or R will most likely default to character for that column.
- It is very important that you become proficient in working with data frames in order to fully understand data analysis.

Example

Consider the following

Product	apple	Banana
price store A	23	56
price store B	67	80

- This is not regarded as a dataframe because here price store is divided into two parts.
- Rearranging the data by taking **product** as one variable and **price** as next variable and **store** as one other variable then it becomes **dataframe**.

Product	Price	Store
apple	23	A
apple	67	B
banana	56	A
banana	80	B

Creating dataframes

- We usually create a dataframe with vectors.
- We can also load dataframes in R using functions like `read.table()` or `read.csv()`
- There are other methods of creating dataframes too

```
names <- c("Angela", "Shondra")
ages <- c(27, 36)
insurance <- c(TRUE, T)
patients <- data.frame(names, ages, insurance)
patients
```

```
##     names ages insurance
## 1  Angela   27      TRUE
## 2 Shondra   36      TRUE
```

- **Ex:** Add a third patient to this list

Adding Rows or Columns

- We may wish to add rows or columns to our data.
- We can do this with:
  - `rbind()`
  - `cbind()`
- For example using our patient data and say we wish to add another patient we could just do the following

```
p.three <- c(names = "Ann Some", age = 45, insurance = TRUI
rbind(patients, p.three)
```

```
##      names ages insurance
## 1   Angela   27      TRUE
## 2  Shondra   36      TRUE
## 3 Ann Some   45      TRUE
```

Adding Rows or Columns

- There may be a warning (in the previous syntax) that serves as a reminder to always know what your `data type` is.
- R may have read your data in as a factor when we want it as a character.

```
patients$names <- as.character(patients$names)
patients <- rbind(patients, p.three)
patients
```

```
##      names ages insurance
## 1   Angela   27      TRUE
## 2  Shondra   36      TRUE
## 3 Ann Some   45      TRUE
```

Adding Rows or Columns

- Finally if we decided to then place another column of data in we could use the followign syntax

```
# Next appointments
next.appt <- c("05/23/2022", "06/14/2022", "08/25/2022")
# Let R know these are dates
next.appt <- as.Date(next.appt, "%m/%d/%Y")
next.appt
```

```
## [1] "2022-05-23" "2022-06-14" "2022-08-25"
```

Adding Rows or Columns

- Using the cbind function, we now have

```
patients <- cbind(patients, next.appt)
patients
```

```
##      names ages insurance  next.appt
## 1   Angela   27      TRUE 2022-05-23
## 2  Shondra   36      TRUE 2022-06-14
## 3 Ann Some   45      TRUE 2022-08-25
```

Attributes of dataframe

- There are four attributes of dataframes
  - length
  - dimension
  - name
  - class
- Lets check whether `patients` is a dataframe

```
str(patients)
```

```
## 'data.frame':    3 obs. of  4 variables:
##  $ names    : chr  "Angela" "Shondra" "Ann Some"
##  $ ages     : chr  "27" "36" "45"
##  $ insurance: chr  "TRUE" "TRUE" "TRUE"
##  $ next.appt: Date, format: "2022-05-23" "2022-06-14" ..
```

Check the attribute of dataframe

```
names(patients)
```

```
## [1] "names"     "ages"      "insurance" "next.appt"
```

```
dim(patients)
```

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

```
length(patients)
```

```
## [1] 4
```

```
# take only one row
head(patients, n = 1)
```

```
##    names ages insurance  next.appt
## 1 Angela   27      TRUE 2022-05-23
```

Accessing dataframes

- In order to best consider accessing of data frames we will use some built in data from R.

```
library(datasets)
titanic <- data.frame(Titanic)
colnames(titanic)  # Variables in Titanic dataframe
```

```
## [1] "Class"    "Sex"      "Age"      "Survived" "Freq"
```

```
# Preview into data
titanic[1:2, ]
```

```
##   Class  Sex   Age Survived Freq
## 1   1st Male Child       No    0
## 2   2nd Male Child       No    0
```

Accessing dataframes

```r
# Preview into data
head(titanic)
```

```
##   Class    Sex   Age Survived Freq
## 1   1st   Male Child       No    0
## 2   2nd   Male Child       No    0
## 3   3rd   Male Child       No   35
## 4  Crew   Male Child       No    0
## 5   1st Female Child       No    0
## 6   2nd Female Child       No    0
```

```r
# Indexing : same as matrices; accessing age information
titanic[, 3]
```

```
##  [1] Child Child Child Child Child Child Child Child Adu
## [13] Adult Adult Adult Adult Child Child Child Child Chi
## [25] Adult Adult Adult Adult Adult Adult Adult Adult
## Levels: Child Adult
```

Other ways of creating dataframes

```r
# method 2
df <- data.frame(test = 1:6, bar = c(T, T, F, F, T, T))
# method 3
x <- c(1, 3, 5, 7, 11)
y <- c("a", "b", "c", "d", "e")
df1 <- data.frame(x = x, y = y)
# does this work?
df2 <- data.frame(x, y)
```

Printing dataframes

```
print(df)
```

```
##   test   bar
## 1    1  TRUE
## 2    2  TRUE
## 3    3 FALSE
## 4    4 FALSE
## 5    5  TRUE
## 6    6  TRUE
```

```
print(df1)
```

```
##    x y
## 1  1 a
## 2  3 b
## 3  5 c
## 4  7 d
## 5 11 e
```

Printing dataframes

```
print(df2)
```

```
##    x y
## 1  1 a
## 2  3 b
## 3  5 c
## 4  7 d
## 5 11 e
```

```
# to checck class and structure of the dataframe
class(df)  #what of df1, df2?
```

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

```
str(df1)  #what of df, df2?
```

```
## 'data.frame':    5 obs. of  2 variables:
##  $ x: num  1 3 5 7 11
##  $ y: chr  "a" "b" "c" "d" ...
```

Creating a dataframe with numerical as well as factor columns

- To create a dataframe with numerical as well as factor columns in R, we simply need to add factor function before factor columns and numerical columns will be created without mentioning any specific characteristics.
- The values of numerical columns just need to be numerical in nature

```
# lets sample 4 teenagers randomly
Sex <- factor(sample(c("Male", "Female"), 4, replace = TRUE
Age <- sample(13:19, 4)
df3 <- data.frame(Sex, Age)
df3
```

```
##      Sex Age
## 1   Male  13
## 2 Female  16
## 3 Female  15
## 4 Female  18
```

Tibbles in R

- Previously we have worked with data in the form of
  - Vectors
  - Lists
  - Matrices
  - Dataframes
- **Tibbles** are a new modern data frame.
- It keeps many important features of the original dataframe.
- It removes many of the outdated features.
- **We will learn more about Tibbles in the coming sessions**

Exercises: dataframes

Exercises: dataframes

Ex. 1

Create the following dataframe, hence invert the variable `Sex` for all individuals.

	Age	Height	Weight	Sex
Alex	25	177	57	F
Lilly	31	163	69	F
Mark	23	190	83	M
Oliver	52	179	75	M
Martha	76	163	70	F
Lucas	49	183	83	M
Caroline	26	164	53	F

Ex. 2

Create this data frame (make sure you import the variable `Working` as character and not factor).

	Working
Alex	Yes
Lilly	No
Mark	No
Oliver	Yes
Martha	Yes
Lucas	No
Caroline	Yes

- Add this data frame column-wise to the previous one.
  a. How many rows and columns does the new data frame have?
  b. What class of data is in each column?

Exercises: dataframes

Ex. 3

Check what class of data is the (built-in data set) state.center and convert it to data frame.

Ex. 4

Create a simple data frame from 3 vectors. Order the entire data frame by the first column.

Ex. 5

Create a data frame from a matrix of your choice, change the row names so every row says id_i (where i is the row number) and change the column names to variable_i (where i is the column number). i.e., for column 1 it will say variable_1, and for row 2 will say id_2 and so on.

Solutions available

RStudio project

Creating an analysis project in RStudio

- Using R Studio to create a Project - From an existing directory
- Go to `File -> New Project` then follow instructions to create new project
- Every time you open the project, it will upload all the files associated with the project.