Exploring data 2

Other R objects: Matrices and lists

Matrices

A matrix is like a data frame, but all the values in all columns must be of the same class (e.g., numeric, character). (Another way you can think of it is as a "wrapped" vector.)

Matrices can be faster and more memory-efficient than data frames. Also, a lot of statistical methods within R code is implemented using linear algebra and other mathematical techniques based on matrices.

Matrices

We can use the `matrix()` function to construct a matrix:

```
foo <- matrix(1:10, ncol = 5)
foo
```

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

Matrices

The `as.matrix()` function is used to convert an object to a matrix:

```
foo <- data.frame(col_1 = 1:2, col_2 = 3:4,
                  col_3 = 5:6, col_4 = 7:8,
                  col_5 = 9:10)
foo <- as.matrix(foo)
foo
```

```
##      col_1 col_2 col_3 col_4 col_5
## [1,]     1     3     5     7     9
## [2,]     2     4     6     8    10
```

Matrices

You can index matrices with square brackets, just like data frames:

```
foo[1, 1:2]
```

```
## col_1 col_2
##     1     3
```

You cannot, however, use `dplyr` functions with matrices:

```
foo %>% filter(col_1 == 1)
```

```
Error in UseMethod("filter_") :
  no applicable method for 'filter_' applied to an object of
  class "c('matrix', 'integer', 'numeric')"
```

Lists

Lists

Lists are the "kitchen sink" of R objects. They can be used to keep together a variety of different R objects of different classes, dimensions, and structures in a single R object.

Because there are often cases where an R operation results in output that doesn't have a simple structure, lists can be a very useful way to output complex output from an R function.

Most lists are not "tidy" data. However, we'll cover some ways that you can easily "tidy" some common list objects you might use a lot in your R code, including the output of fitting linear and generalized linear models.

Lists

```
example_list <- list(a = sample(1:10, 5),
                     b = tibble(letters = letters[1:3],
                                numbers = 1:3))
example_list
```

```
## $a
## [1]  9 10  1  2  7
##
## $b
## # A tibble: 3 x 2
##   letters numbers
##   <chr>     <int>
## 1 a             1
## 2 b             2
## 3 c             3
```

Indexing lists

To pull an element out of a list, you can either use $ or [[]] indexing:

```
example_list$a
```

```
## [1]  9 10  1  2  7
```

```
example_list[[2]]
```

```
## # A tibble: 3 x 2
##   letters numbers
##   <chr>     <int>
## 1 a             1
## 2 b             2
## 3 c             3
```

Indexing lists

To access a specific value within a list element we can index the element using double, double brackets:

```
example_list[["b"]][["numbers"]]
```

```
## [1] 1 2 3
```

Again, we can index using names or numeric indices:

```
example_list[["b"]][[1]]
```

```
## [1] "a" "b" "c"
```

Exploring lists

If an R object is a list, running `class` on the object will return "list":

```
class(example_list)
```

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

Often, lists will have names for each element (similar to column names for a dataframe). You can get the names of all elements of a list using the `names` function:

```
names(example_list)
```

```
## [1] "a" "b"
```

Exploring lists

The `str` function is also useful for exploring the structure of a list object:

```
str(example_list)
```

```
## List of 2
##  $ a: int [1:5] 9 10 1 2 7
##  $ b: tibble [3 x 2] (S3: tbl_df/tbl/data.frame)
##   ..$ letters: chr [1:3] "a" "b" "c"
##   ..$ numbers: int [1:3] 1 2 3
```

Exploring lists

A list can even contain other lists. We can use the `str` function to see the structure of a list:

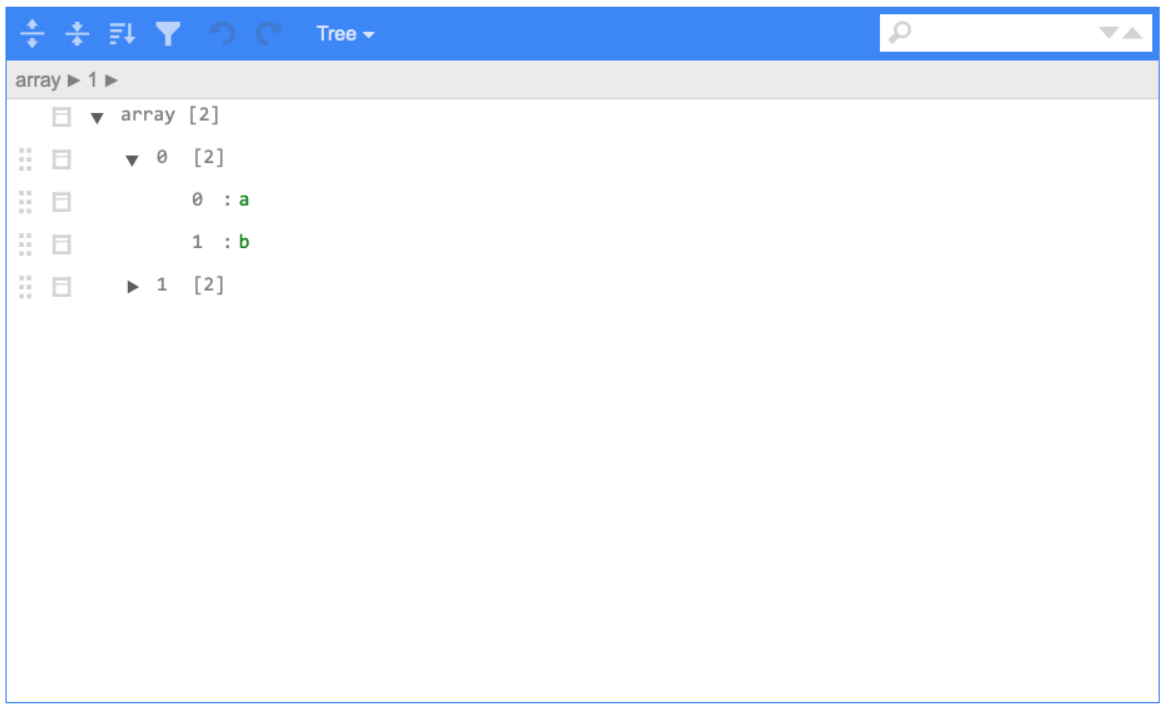
```
a_list <- list(list("a", "b"), list(1, 2))

str(a_list)
```

```
## List of 2
##  $ :List of 2
##   ..$ : chr "a"
##   ..$ : chr "b"
##  $ :List of 2
##   ..$ : num 1
##   ..$ : num 2
```

Exploring lists

Using `str` to print out the list's structure doesn't produce the easiest to digest output. We can use the `jsonedit` function from the `listviewer` package to create a widget in the Viewer pane to more esily explore our list.

```
library(listviewer)
jsonedit(a_list)
```



Lists versus dataframes

As a note, a dataframe is actually just a very special type of list. It is a list where every element (column in the dataframe) is a vector of the same length, and the object has a special attribute specifying that it is a dataframe.

```
example_df <- tibble(letters = letters[1:3],
                     number = 1:3)
class(example_df)
```

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

```
is.list(example_df)
```

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