Lab 3 - Data Types | Exploring Datasets

Environmental Data Analytics | John Fay and Luana Lima

Spring 2023

Objectives

1. Discuss and navigate different data types in R
2. Create, manipulate, and explore datasets
3. Date objects

Data Types in R

R treats objects differently based on their characteristics. For more information, please see: https://www.statmethods.net/input/datatypes.html.

- **Vectors** 1 dimensional structure that contains elements of the same type.
- **Matrices** 2 dimensional structure that contains elements of the same type.
- **Arrays** Similar to matrices, but can have more than 2 dimensions. We will not delve into arrays in depth.
- **Lists** Ordered collection of elements that can have different modes.
- **Data Frames** 2 dimensional structure that is more general than a matrix. Columns can have different modes (e.g., numeric and factor). When we import csv files into the R workspace, they will enter as data frames.

Define what each new piece of syntax does below (i.e., fill in blank comments). Note that the R chunk has been divided into sections (# at beginning of line, —- at end)

```
# Vectors ----
vector1 <- c(1,2,5.3,6,-2,4) # numeric vector
vector1
```

```
## [1]  1.0  2.0  5.3  6.0 -2.0  4.0
```

```
vector2 <- c("one","two","three") # character vector
vector2
```

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

```
vector3 <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) #logical vector
vector3
```

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

```
vector1[3] #
```

```
## [1] 5.3
```

```
# Matrices ----
matrix1 <- matrix(1:20, nrow = 5,ncol = 4) #
matrix1
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
## [4,]    4    9   14   19
## [5,]    5   10   15   20
```

```
matrix2 <- matrix(1:20, nrow = 5, ncol = 4, byrow = TRUE) #
matrix2
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]   13   14   15   16
## [5,]   17   18   19   20
```

```
matrix3 <- matrix(1:20, nrow = 5, ncol = 4, byrow = TRUE, # return after comma continues the line
                  dimnames = list(c("uno", "dos", "tres", "quatro", "cinco"),
                                  c("un", "deux", "trois", "quatre"))) #

matrix1[4, ] # [row number 4, no column number]
```

```
## [1]  4  9 14 19
```

```
matrix1[ , 3] # [no row number, column number 3]
```

```
## [1] 11 12 13 14 15
```

```
matrix1[c(12, 14)] # [value 12, value 14]
```

```
## [1] 12 14
```

```
matrix1[c(12:14)] # [value 12 through value 14]
```

```
## [1] 12 13 14
```

```
matrix1[2:4, 1:3] # [row 2-4, column 1-3] But the output renumbers the row and column back to 1 (not th
```

```
##      [,1] [,2] [,3]
## [1,]    2    7   12
## [2,]    3    8   13
## [3,]    4    9   14
```

```
cells <- c(1, 26, 24, 68)
rnames <- c("R1", "R2")
cnames <- c("C1", "C2")
matrix4 <- matrix(cells, nrow = 2, ncol = 2, byrow = TRUE, #default is by column?
  dimnames = list(rnames, cnames)) # default seems to put row names first, then column.
matrix4
```

```
##    C1 C2
## R1  1 26
## R2 24 68
```

```
test <- c(1, 26, 24, 68)
rnames <- c("R1", "R2")
cnames <- c("C1", "C2")
test_matrix <- matrix(cells, nrow = 2, ncol = 2,
  dimnames = list(rnames, cnames)) # default seems to put row names first, then column.
test_matrix
```

```
##    C1 C2
## R1  1 24
## R2 26 68
```

```
# Lists ----
list1 <- list(name = "Maria", mynumbers = vector1, mymatrix = matrix1, age = 5.3); list1
```

```
## $name
## [1] "Maria"
##
## $mynumbers
## [1]  1.0  2.0  5.3  6.0 -2.0  4.0
##
## $mymatrix
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
## [4,]    4    9   14   19
## [5,]    5   10   15   20
##
## $age
## [1] 5.3
```

```
list1[[2]]
```

```
## [1]  1.0  2.0  5.3  6.0 -2.0  4.0
```

```
# Data Frames ---- column default
d <- c(1, 2, 3, 4) # What type of vector? -numeric
e <- c("red", "white", "red", NA) # What type of vector? -character
f <- c(TRUE, TRUE, TRUE, FALSE) # What type of vector? -logical
dataframe1 <- data.frame(d,e,f) #
names(dataframe1) <- c("ID","Color","Passed"); View(dataframe1) #


dataframe1[1:2,] #
```

```
##   ID Color Passed
## 1  1   red   TRUE
## 2  2 white   TRUE
```

```
dataframe1[c("ID","Passed")] #
```

```
##   ID Passed
## 1  1   TRUE
## 2  2   TRUE
## 3  3   TRUE
## 4  4  FALSE
```

```
dataframe1$ID
```

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

Question: How do the different types of data appear in the Environment tab?

> Answer:

Question: In the R chunk below, write "dataframe1$". Press `tab` after you type the dollar sign. What happens?

> Answer:

Coding challenge

Find a ten-day forecast of temperatures (Fahrenheit) for Durham, North Carolina. Create two vectors, one representing the high temperature on each of the ten days and one representing the low.

Now, create two additional vectors that include the ten-day forecast for the high and low temperatures in Celsius. Use a function to create the two new vectors from your existing ones in Fahrenheit.

Combine your four vectors into a data frame and add informative column names.

Use the common functions `summary` and `sd` to obtain basic data summaries of the ten-day forecast. How would you call these functions differently for the entire data frame vs. a single column? Attempt to demonstrate both options below.

Date objects

Remember formatting of dates in R:

%d day as number (0-31) %m month (00-12, can be e.g., 01 or 1) %y 2-digit year %Y 4-digit year %a abbreviated weekday %A unabbreviated weekday %b abbreviated month %B unabbreviated month

```
# Adjust date formatting for today Write code for three different date
# formats.  An example is provided to get you started.  (code must be
# uncommented)
today <- Sys.Date()
format(today, format = "%B")
```

```
## [1] "March"
```

```
# format(today, format = '') format(today, format = '') format(today,
# format = '')
```

Package lubridate

Install and load the package lubridate into your R session. Lubridate offers fast and user friendly parsing of date-time data. Create a string for today's data and then convert it to R date object using lubridate.

More info on lubridate [here][https://cran.r-project.org/web/packages/lubridate/lubridate.pdf].

```
# install.packages('lubridate')
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```r
# Ex1
str_today <- "2023-feb-7"
# Since the format is year-month-day we will use function ymd()
date_obj_today <- ymd(str_today)
date_obj_today
```

```
## [1] "2023-02-07"
```

```r
# Ex2
str_today2 <- "Feb 7, 2023"
# Since the format is month-day-year we will use function mdy()
date_obj_today2 <- mdy(str_today2)
date_obj_today2
```

```
## [1] "2023-02-07"
```

```r
# Ex_3 - on your own...
str_juneteenth <- "19 June 1865"
# Since the format is month-day-year we will use function mdy()? --Isn't
# it dmy?
date_juneteenth <- dmy(str_juneteenth)
date_juneteenth
```

```
## [1] "1865-06-19"
```

```r
# century issue
str_past <- "55-feb-3"
date_obj_past <- ymd(str_past)
date_obj_past
```

```
## [1] "2055-02-03"
```

```r
# Build a function to fix year that is more general than the one discussed
# in the lesson
fix.early.dates <- function(d, cutoff) {
    m <- year(d)%%100  #operator %% is a modular division i.e. integer-divide year(d) by 100 and return
    year(d) <- ifelse(m > cutoff, 1900 + m, 2000 + m)  #this will update year(d), year() is a function
    return(d)
}

fixed_date_obj_past <- fix.early.dates(date_obj_past, cutoff = 23)  #cutoff could be the current year t
fixed_date_obj_past
```

```
## [1] "1955-02-03"
```

```r
test_date <- "89-feb-3"
test_date_conv <- ymd(test_date)
test_date_conv
```

```
## [1] "1989-02-03"
```

```r
fixed_test_date <- fix.early.dates(test_date_conv, cutoff = 23)
```

```r
# Fix for century issue
str_past <- "55-feb-3"
# Alternative 1
date_obj_past <- fast_strptime(str_past, "%y-%b-%d", cutoff_2000 = 23L)
date_obj_past
```

```
## [1] "1955-02-03 UTC"
```

```
# Alternative 2
date_obj_past2 <- parse_date_time2(str_past, "ymd", cutoff_2000 = 23L)
date_obj_past2
```

```
## [1] "1955-02-03 UTC"
```

```
# Functions ymd(), mdy(), ydm() do not take argument cutoff_2000
```

In some cases when dates are provided as integers, you may need to provide an origin for your dates. For example, excel date could be given as number of days since an origin date. Origin date can be different. When R looks at dates as integers, its origin is January 1, 1970. Check if that is true on your machine.

```
# Check if '1970-01-01' is your origin date.
lubridate::origin
```

```
## [1] "1970-01-01 UTC"
```