Intro to R

Data Classes

Data Types:

- One dimensional types (“vectors”):
  - Character: strings or individual characters, quoted
  - Numeric: any real number(s)
  - Integer: any integer(s)/whole numbers
  - Factor: categorical/qualitative variables
  - Logical: variables composed of TRUE or FALSE
  - Date/POSIXct: represents calendar dates and times

Character and numeric

We have already covered `character` and `numeric` types.

```
class(c("Andrew", "Jaffe"))
```

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

```
class(c(1, 4, 7))
```

```
## [1] "numeric"
```

Integer

`Integer` is a special subset of `numeric` that contains only whole numbers

A sequence of numbers is an example of the integer type

```
x = seq(from = 1, to = 5) # seq() is a function
x
```

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

```
class(x)
```

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

Logical

`logical` is a type that only has two possible elements: `TRUE` and `FALSE`

```
x = c(TRUE, FALSE, TRUE, TRUE, FALSE)
class(x)
```

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

Note that `logical` elements are NOT in quotes.

```
z = c("TRUE", "FALSE", "TRUE", "FALSE")
class(z)
```

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

General Class Information

There are two useful functions associated with practically all R classes, which relate to logically checking the underlying class (`is.CLASS_()`) and coercing between classes (`as.CLASS_()`).

```
is.numeric(c("Andrew", "Jaffe"))
```

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

```
is.character(c("Andrew", "Jaffe"))
```

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

General Class Information

There are two useful functions associated with practically all R classes:

- `is.CLASS_NAME(x)` to check whether or not `x` is of certain class
- `as.CLASS_NAME(x)` to coerce `x` from current `x` class into a certain class

```
is.character(c(1, 4, 7))
```

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

```
is.numeric(c(1, 4, 7))
```

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

General Class Information: coercing

In some cases the coercing is seamless

```
as.character(c(1, 4, 7))
```

```
## [1] "1" "4" "7"
```

```
as.numeric(c("1", "4", "7"))
```

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

```
as.logical(c("TRUE", "FALSE", "FALSE"))
```

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

General Class Information: coercing

In some cases the coercing is not possible; if executed, will return `NA` (an R constant representing "**N**ot **A**vailable" i.e. missing value)

```
as.numeric(c("1", "4", "7a"))
```

```
## Warning: NAs introduced by coercion
```

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

```
as.logical(c("TRUE", "FALSE", "UNKNOWN"))
```

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

```
as.Date(c("2021-06-15", "2021-06-32"))
```

```
## [1] "2021-06-15" NA
```

Factors

A `factor` is a special `character` vector where the elements have pre-defined groups or 'levels'. You can think of these as qualitative or categorical variables:

```
x <- c("red", "red", "blue", "yellow", "blue")
class(x)
```

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

```
x_fact = factor(x)  # factor() is a function
class(x_fact)
```

```
## [1] "factor"
```

```
x_fact
```

```
## [1] red    red    blue   yellow blue
## Levels: blue red yellow
```

Note that levels are, by default, in alphanumerical order.

Factors

You can learn what are the unique levels of a `factor` vector

```
levels(x_fact)
```

```
## [1] "blue"   "red"    "yellow"
```

To change the levels ordering, use `relevel()` function.

Factors

Factors can be converted to `numeric` or `character` very easily

```
x_fact
```

```
## [1] red    red    blue   yellow blue
## Levels: blue red yellow
```

```
as.character(x_fact)
```

```
## [1] "red"    "red"    "blue"   "yellow" "blue"
```

```
as.numeric(x_fact)
```

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

Factors

Note that R:

- reads in character strings as `factor` class by default for some functions like `read.csv()` from base R
- reads in character strings as `character` class by default for other functions like `read_csv()` from `readr` package

Useful functions to create vectors

For character: `rep()`

```
rep(c("black", "white"), each = 3)
```

```
## [1] "black" "black" "black" "white" "white" "white"
```

```
rep(c("black", "white"), times = 3)
```

```
## [1] "black" "white" "black" "white" "black" "white"
```

Useful functions to create vectors

For numeric: `seq()`

```
seq(from = 0, to = 1, by = 0.2)
```

```
## [1] 0.0 0.2 0.4 0.6 0.8 1.0
```

```
seq(from = -5, to = 5, length.out = 10)
```

```
##  [1] -5.0000000 -3.8888889 -2.7777778 -1.6666667 -0.5555556  0.5555556
##  [7]  1.6666667  2.7777778  3.8888889  5.0000000
```

Lab Part 1

Lab document:
http://jhudatascience.org//intro_to_r/Data_Classes/lab/Data_Classes_Lab.Rmd

Dates

There are two most popular R classes used when working with dates and times:

- `Date` class representing a calendar date
- `POSIXct` class representing a calendar date with hours, minutes, seconds

We convert data from character to `Date`/`POSIXct` to use functions to manipilate date/date and time

`lubridate` is a powerful, widely used R package from “tidyverse” family to work with `Date` / `POSIXct` class objects

Creating `Date` class object

```
class("2021-06-15")
```

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

```
as.Date("2021-06-15")          # base R
```

```
## [1] "2021-06-15"
```

```
class(as.Date("2021-06-15"))   # base R
```

```
## [1] "Date"
```

Creating `Date` class object

```
class("2021-06-15")
```

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

```
library(lubridate)

ymd("2021-06-15")                # lubridate package
```

```
## [1] "2021-06-15"
```

```
class(ymd("2021-06-15"))         # lubridate package
```

```
## [1] "Date"
```

Note for function `ymd`: **y**year **m**onth **d**ay

Creating `Date` class object

```
mdy("06/15/2021")
```

```
## [1] "2021-06-15"
```

```
mdy("06/15/21")
```

```
## [1] "2021-06-15"
```

Note for function `mdy`: **m**onth **d**ay **y**year

Creating `POSIXct` class object

```
class("2013-01-24 19:39:07")
```

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

```
ymd_hms("2013-01-24 19:39:07")              # lubridate package
```

```
## [1] "2013-01-24 19:39:07 UTC"
```

```
class(ymd_hms("2013-01-24 19:39:07"))       # lubridate package
```

```
## [1] "POSIXct" "POSIXt"
```

UTC represents time zone, by default: Coordinated Universal Time

Note for function `ymd_hms`: **y**year **m**onth **d**ay **h**our **m**inute **s**econd.

There are functions in case your data have only date, hour and minute (`ymd_hm()`) or only date and hour (`ymd_h()`).

Some useful functions from `lubridate` to manipulate `Date` objects

```
x <- ymd(c("2021-06-15", "2021-07-15"))
x
```

```
## [1] "2021-06-15" "2021-07-15"
```

```
day(x)    # see also: month(x) , year(x)
```

```
## [1] 15 15
```

```
x + days(10)
```

```
## [1] "2021-06-25" "2021-07-25"
```

```
x + months(1) + days(10)
```

```
## [1] "2021-07-25" "2021-08-25"
```

```
wday(x, label = TRUE)
```

```
## [1] Tue Thu
## Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat
```

Some useful functions from `lubridate` to manipulate `POSIXct` objects

```
x <- ymd_hms("2013-01-24 19:39:07")
x
```

```
## [1] "2013-01-24 19:39:07 UTC"
```

```
date(x)
```

```
## [1] "2013-01-24"
```

```
x + hours(3)
```

```
## [1] "2013-01-24 22:39:07 UTC"
```

```
floor_date(x, "1 hour")  # see also: ceiling_date()
```

```
## [1] "2013-01-24 19:00:00 UTC"
```

Differences in dates

```
x1 <- ymd(c("2021-06-15"))
x2 <- ymd(c("2021-07-15"))

difftime(x2, x1, units = "weeks")
```

```
## Time difference of 4.285714 weeks
```

```
as.numeric(difftime(x2, x1, units = "weeks"))
```

```
## [1] 4.285714
```

Similar can be done with time (e.g. difference in hours).

Lab Part 2

Lab document:
http://jhudatascience.org//intro_to_r/Data_Classes/lab/Data_Classes_Lab.Rmd

Two-dimensional data classes

Two-dimensional classes are those we would often use to store data read from a file

- a data frame (`data.frame` or `tibble` class)
  - "traditional", Excel-like spreadsheets
  - different columns (variables) can be of different classes
  - for example one variable is calendar date – `Date` class, another variable is age – `numeric` class
- a matrix (`matrix` class)
  - also composed of rows and columns
  - unlike data frame, the entire matrix is composed of one R class
  - for example: all entries are `numeric`, or all entries are `character`

Matrices

```
n = 1:9
n
```

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

```
mat = matrix(n, nrow = 3)
mat
```

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

Matrices: data selection

Note you cannot use `dplyr` functions (like `select`) on matrices. To subset matrix rows and/or columns, use `matrix[row_index, column_index]`.

```
mat[1, 1] # individual entry: row 1, column 1
```

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

```
mat[1, ]  # first row
```

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

```
mat[, 1]  # first column
```

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

```
mat[c(1,2), c(2,3)]   # subset of original matrix: two rows and two columns
```

```
##      [,1] [,2]
## [1,]    4    7
## [2,]    5    8
```

Lists

- One other data type that is the most generic are `lists
- Can be created using `list()`
- Can hold vectors, strings, matrices, models, list of other list!

```
mylist <- list(c("A", "b", "c"), c(1,2,3), matrix(1:4, ncol = 2))
mylist
```

```
## [[1]]
## [1] "A" "b" "c"
##
## [[2]]
## [1] 1 2 3
##
## [[3]]
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
class(mylist)
```

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

Lists

List elements can be named

```
mylist_named <- list(letters = c("A", "b", "c"),
                     numbers = c(1,2,3),
                     one_matrix = matrix(1:4, ncol = 2))
mylist_named
```

```
## $letters
## [1] "A" "b" "c"
##
## $numbers
## [1] 1 2 3
##
## $one_matrix
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

Lists: data selection

You can reference data from list using `$` (if elements are named) or using `[[ ]]`

```
mylist_named[[1]]
```

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

```
mylist_named[["letters"]]     # works only for a list with elements' names
```

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

```
mylist_named$letters          # works only for a list with elements' names
```

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

Lab Part 3

Lab document:
http://jhudatascience.org//intro_to_r/Data_Classes/lab/Data_Classes_Lab.Rmd