Data Classes

One dimensional types (“vectors”)

Data Types

* 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("tree", "cloud", "stars_&_sky"))
```

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

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

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

Character and numeric

This can also be a bit tricky.

```
class(c(1, 2, "tree"))
```

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

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

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

Numeric Subclasses

There are two major numeric subclasses

1. Integer
2. Double

Integer

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

```
x <- c(1, 2, 3, 4, 5)
class(x)
```

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

```
typeof(x)
```

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

Double

`Double` is a special subset of `numeric` that contains fractional values.

`Double` stands for double-precision

```
y <- c(1.1, 2.0, 3.2, 4.5, 5.6)
y
```

```
## [1] 1.1 2.0 3.2 4.5 5.6
```

```
class(y)
```

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

```
typeof(y)
```

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

Checking double vs integer

A `tibble` will show the difference (as does `glimpse()`)

```
tibble(xvar = x, yvar = y)
```

```
## # A tibble: 5 × 2
##    xvar  yvar
##   <dbl> <dbl>
## 1     1   1.1
## 2     2   2
## 3     3   3.2
## 4     4   4.5
## 5     5   5.6
```

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:

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

General Class Information: Checking

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

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

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

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

```
is.character(c("tree", "cloud"))
```

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

```
is.numeric(c("tree", "cloud"))
```

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

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
```

```
as.integer(c(1.2, 3.7))
```

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

```
as.double(c(1, 2, 3))
```

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

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. Use the `factor()` function to create factors.

```
x <- c("small", "mediam", "large", "medium", "large")
class(x)
```

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

```
x_fact <- factor(x) # factor() is a function
class(x_fact)
```

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

```
x_fact
```

```
## [1] small  mediam large  medium large
## Levels: large mediam medium small
```

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] "large"  "mediam" "medium" "small"
```

More on how to change the levels ordering in a lecture coming up!

Factors

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

```
x_fact
```

```
## [1] small  mediam large  medium large
## Levels: large mediam medium small
```

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

```
## [1] "small"  "mediam" "large"  "medium" "large"
```

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

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

Useful functions to create vectors `rep()`

For character: `rep()` can create very long vectors. Works for creating character and numeric vectors.

The `each` argument specifies how many of each item you want repeated. The `times` argument specifies how many times you want the vector repeated.

```
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"
```

```
rep(c("black", "white"), each = 2, times = 2)
```

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

Useful functions to create vectors **seq()**

For numeric: `seq()` can be very useful- both integer and double.
The `from` argument says what number to start on.
The `to` argument says what number to not go above.
The `by` argument says how much to increment by.
The `length.out` argument says how long the vector should be overall.

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

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

```
seq(from = 0, to = 10, by = 1)
```

```
##  [1]  0  1  2  3  4  5  6  7  8  9 10
```

```
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
```

Creating numeric vectors `sample()`

You can use the `sample()` function to make a random sequence. The `x` argument specifies what you are sampling from. The `size` argument specifies how many values there should be. The `replace` argument specifies if values should be replaced or not.

```
seq_hun <- seq(from = 0, to = 100, by = 1)
seq_hun
```

```
##   [1]   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [19]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
##  [37]  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
##  [55]  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
##  [73]  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
##  [91]  90  91  92  93  94  95  96  97  98  99 100
```

```
y <- sample(x = seq_hun, size = 5, replace = TRUE)
y
```

```
## [1] 19 40 62 70 60
```

Summary

- There are two types of numeric class objects: integer and double
- Logic class objects only have `TRUE` or `False` (without quotes)
- is.CLASS_NAME(x) can be used to test the class of an object x
- as.CLASS_NAME(x) can be used to change the class of an object x
- Factors are a special character class that has levels - more on that soon!
- The repeat `rep()` function helps you create vectors with the `to` and `from` arguments
- The sequence `seq()` function helps you create numeric vectors (`from`,`to`, `by`, and `length.out` arguments)
- `seq()` can be used for integers or double numeric vectors (`by` argument)
- `sample()` makes random vectors. Can be used for integers or double depending on what it is sampling from.
- tibbles show column classes!

Lab Part 1

- Class Website
- Lab

Two-dimensional data classes

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)
- a matrix (`matrix` class)
  - also composed of rows and columns
  - unlike `data.frame` or `tibble`, the entire matrix is composed of one R class
  - for example: all entries are `numeric`, or all entries are `character`

Matrices

`as.matrix()` creates a matrix from a data frame (where all values are the same class). `matrix()` creates a matrix from scratch.

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
class(iris)
```

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

```
iris_mat <- head(tibble(select(iris, -Species)))
as.matrix(iris_mat)
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## [1,]          5.1         3.5          1.4         0.2
## [2,]          4.9         3.0          1.4         0.2
## [3,]          4.7         3.2          1.3         0.2
## [4,]          4.6         3.1          1.5         0.2
## [5,]          5.0         3.6          1.4         0.2
## [6,]          5.4         3.9          1.7         0.4
```

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
```

Special data classes

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 manipulate 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"
```

```
library(lubridate)
```

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

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

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

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

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

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

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

dates

```
a <- ymd("2021-06-15")
b <- ymd("2021-06-18")
a - b
```

```
## Time difference of -3 days
```

Creating `Date` class object

This will not work: `date()` is picky…

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

This works though!

```
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()`).

Summary

- two dimensional object classes include: data frames, tibbles, matrices, and lists
- matrix has columns and rows but is all one data class
  - can create a matrix with `matrix()` from scratch or `as.matrix()` from something
- lists can contain multiples of any other class of data including lists!
  - can create lists with `list()`
- calendar dates can be represented with the `Date` class using `ymd()`, `mdy()` functions from `lubridate` package
- `POSIXct` class representing a calendar date with hours, minutes, seconds. Can use `ymd_hms()` or `ymd_hm()` or `ymd_h()`functions from the `lubridate` package

Lab Part 2

- Class Website
- Lab

Extra Slides

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).

Data Selection

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
```

Vectors: data selection

To get element(s) of a vector (one-dimensional object):

- Type the name of the variable and open the rectangular brackets `[ ]`
- In the rectangular brackets, type index (/vector of indexes) of element (/elements) you want to pull. **In R, indexes start from 1** (not: 0)

```
x <- c("a", "b", "c", "d", "e", "f", "g", "h")
x
```

```
## [1] "a" "b" "c" "d" "e" "f" "g" "h"
```

```
x[2]
```

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

```
x[c(1, 2, 100)]
```

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

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

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

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

```
mat[1, 2] # individual entry: row 1, column 2
```

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

```
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
```

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"
```