Data types

Data Science in a Box

datasciencebox.org

Why should you care about data types?

Example: Cat lovers

A survey asked respondents their name and number of cats. The instructions said to enter the number of cats as a numerical value.

```
cat_lovers <- read_csv("data/cat-lovers.csv")
```

```
## # A tibble: 60 x 3
##   name           number_of_cats handedness
##   <chr>          <chr>          <chr>
## 1 Bernice Warren 0              left
## 2 Woodrow Stone  0              left
## 3 Willie Bass    1              left
## 4 Tyrone Estrada 3              left
## 5 Alex Daniels   3              left
## 6 Jane Bates     2              left
## # ... with 54 more rows
```

Oh why won't you work?!

```
cat_lovers %>%
  summarise(mean_cats = mean(number_of_cats))
```

```
## Warning in mean.default(number_of_cats): argument is not numeric
## or logical: returning NA
```

```
## # A tibble: 1 x 1
##   mean_cats
##       <dbl>
## 1        NA
```

```
?mean
```

mean {base} R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

`x`	An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.
`trim`	the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.
`...`	further arguments passed to or from other methods.

Oh why won't you still work??!!

```
cat_lovers %>%
  summarise(mean_cats = mean(number_of_cats, na.rm = TRUE))
```

```
## Warning in mean.default(number_of_cats, na.rm = TRUE): argument
## is not numeric or logical: returning NA
```

```
## # A tibble: 1 x 1
##   mean_cats
##       <dbl>
## 1        NA
```

Take a breath and look at your data

> What is the type of the `number_of_cats` variable?

```r
glimpse(cat_lovers)
```

```
## Rows: 60
## Columns: 3
## $ name           <chr> "Bernice Warren", "Woodrow Stone", "Will~
## $ number_of_cats <chr> "0", "0", "1", "3", "3", "2", "1", "1", ~
## $ handedness     <chr> "left", "left", "left", "left", "left", ~
```

Let's take another look

Show 10 entries Search:

	name	number_of_cats	handedness
1	Bernice Warren	0	left
2	Woodrow Stone	0	left
3	Willie Bass	1	left
4	Tyrone Estrada	3	left
5	Alex Daniels	3	left
6	Jane Bates	2	left
7	Latoya Simpson	1	left
8	Darin Woods	1	left
9	Agnes Cobb	0	left
10	Tabitha Grant	0	left

Showing 1 to 10 of 60 entries Previous 1 2 3 4 5 6 Next

Sometimes you might need to babysit your respondents

```
cat_lovers %>%
  mutate(number_of_cats = case_when(
    name == "Ginger Clark" ~ 2,
    name == "Doug Bass"    ~ 3,
    TRUE                   ~ as.numeric(number_of_cats)
    )) %>%
  summarise(mean_cats = mean(number_of_cats))
```

```
## Warning in eval_tidy(pair$rhs, env = default_env): NAs introduced
## by coercion

## # A tibble: 1 x 1
##   mean_cats
##       <dbl>
## 1     0.833
```

Always you need to respect data types

```
cat_lovers %>%
  mutate(
    number_of_cats = case_when(
      name == "Ginger Clark" ~ "2",
      name == "Doug Bass"    ~ "3",
      TRUE                   ~ number_of_cats
      ),
    number_of_cats = as.numeric(number_of_cats)
    ) %>%
  summarise(mean_cats = mean(number_of_cats))
```

```
## # A tibble: 1 x 1
##   mean_cats
##       <dbl>
## 1     0.833
```

Now that we know what we're doing...

```r
cat_lovers <- cat_lovers %>%
  mutate(
    number_of_cats = case_when(
      name == "Ginger Clark" ~ "2",
      name == "Doug Bass"    ~ "3",
      TRUE                   ~ number_of_cats
      ),
    number_of_cats = as.numeric(number_of_cats)
    )
```

Moral of the story

- If your data does not behave how you expect it to, type coercion upon reading in the data might be the reason.
- Go in and investigate your data, apply the fix, *save your data*, live happily ever after.

now that we have a good motivation for learning about data types in R

let's learn about data types in R!

Data types

Data types in R

- **logical**
- **double**
- **integer**
- **character**
- and some more, but we won't be focusing on those

Logical & character

logical - boolean values TRUE and FALSE

```
typeof(TRUE)
```

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

character - character strings

```
typeof("hello")
```

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

Double & integer

double - floating point numerical values (default numerical type)

```r
typeof(1.335)
```

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

```r
typeof(7)
```

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

integer - integer numerical values (indicated with an `L`)

```r
typeof(7L)
```

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

```r
typeof(1:3)
```

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

Concatenation

Vectors can be constructed using the `c()` function.

```
c(1, 2, 3)
```

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

```
c("Hello", "World!")
```

```
## [1] "Hello"  "World!"
```

```
c(c("hi", "hello"), c("bye", "jello"))
```

```
## [1] "hi"    "hello" "bye"   "jello"
```

Converting between types

with intention...

```
x <- 1:3
x
```

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

```
typeof(x)
```

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

Converting between types

with intention...

```r
x <- 1:3
x
```

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

```r
y <- as.character(x)
y
```

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

```r
typeof(x)
```

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

```r
typeof(y)
```

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

Converting between types

with intention...

```
x <- c(TRUE, FALSE)
x
```

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

```
typeof(x)
```

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

Converting between types

with intention...

```r
x <- c(TRUE, FALSE)
x
```

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

```r
typeof(x)
```

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

```r
y <- as.numeric(x)
y
```

```
## [1] 1 0
```

```r
typeof(y)
```

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

Converting between types

without intention...

R will happily convert between various types without complaint when different types of data are concatenated in a vector, and that's not always a great thing!

```
c(1, "Hello")
```

```
## [1] "1"     "Hello"
```

```
c(FALSE, 3L)
```

```
## [1] 0 3
```

Converting between types

without intention...

R will happily convert between various types without complaint when different types of data are concatenated in a vector, and that's not always a great thing!

```r
c(1, "Hello")
```

```
## [1] "1"     "Hello"
```

```r
c(FALSE, 3L)
```

```
## [1] 0 3
```

```r
c(1.2, 3L)
```

```
## [1] 1.2 3.0
```

```r
c(2L, "two")
```

```
## [1] "2"   "two"
```

Explicit vs. implicit coercion

Let's give formal names to what we've seen so far:

Explicit vs. implicit coercion

Let's give formal names to what we've seen so far:

- **Explicit coercion** is when you call a function like `as.logical()`, `as.numeric()`, `as.integer()`, `as.double()`, or `as.character()`

Explicit vs. implicit coercion

Let's give formal names to what we've seen so far:

- **Explicit coercion** is when you call a function like `as.logical()`, `as.numeric()`, `as.integer()`, `as.double()`, or `as.character()`
- **Implicit coercion** happens when you use a vector in a specific context that expects a certain type of vector

Your turn!

- RStudio Cloud > `AE 05 - Hotels + Data types` > open `type-coercion.Rmd` and knit.
- What is the type of the given vectors? First, guess. Then, try it out in R. If your guess was correct, great! If not, discuss why they have that type.

Your turn!

- RStudio Cloud > `AE 05 - Hotels + Data types` > open `type-coercion.Rmd` and knit.
- What is the type of the given vectors? First, guess. Then, try it out in R. If your guess was correct, great! If not, discuss why they have that type.

Example: Suppose we want to know the type of `c(1, "a")`. First, I'd look at:

```
typeof(1)
```

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

```
typeof("a")
```

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

and make a guess based on these. Then finally I'd check:

```
typeof(c(1, "a"))
```

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

Special values

Special values

- `NA`: Not available
- `NaN`: Not a number
- `Inf`: Positive infinity
- `-Inf`: Negative infinity

Special values

- `NA`: Not available
- `NaN`: Not a number
- `Inf`: Positive infinity
- `-Inf`: Negative infinity

```
pi / 0
```

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

```
0 / 0
```

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

```
1/0 - 1/0
```

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

```
1/0 + 1/0
```

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

NAs are special ❄️ s

```r
x <- c(1, 2, 3, 4, NA)
```

```r
mean(x)
```

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

```r
mean(x, na.rm = TRUE)
```

```
## [1] 2.5
```

```r
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
##    1.00    1.75    2.50    2.50    3.25    4.00       1
```

NAs are logical

R uses NA to represent missing values in its data structures.

```
typeof(NA)
```

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

Mental model for NAs

- Unlike NaN, NAs are genuinely unknown values
- But that doesn't mean they can't function in a logical way
- Let's think about why NAs are logical...

Mental model for NAs

- Unlike NaN, NAs are genuinely unknown values
- But that doesn't mean they can't function in a logical way
- Let's think about why NAs are logical...

> Why do the following give different answers?

```r
# TRUE or NA
TRUE | NA
```

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

```r
# FALSE or NA
FALSE | NA
```

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

→ See next slide for answers...

- NA is unknown, so it could be TRUE or FALSE

- TRUE | NA

```
TRUE | TRUE   # if NA was TRUE
```

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

```
TRUE | FALSE # if NA was FALSE
```

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

- FALSE | NA

```
FALSE | TRUE   # if NA was TRUE
```

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

```
FALSE | FALSE # if NA was FALSE
```

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

- Doesn't make sense for mathematical operations
- Makes sense in the context of missing data