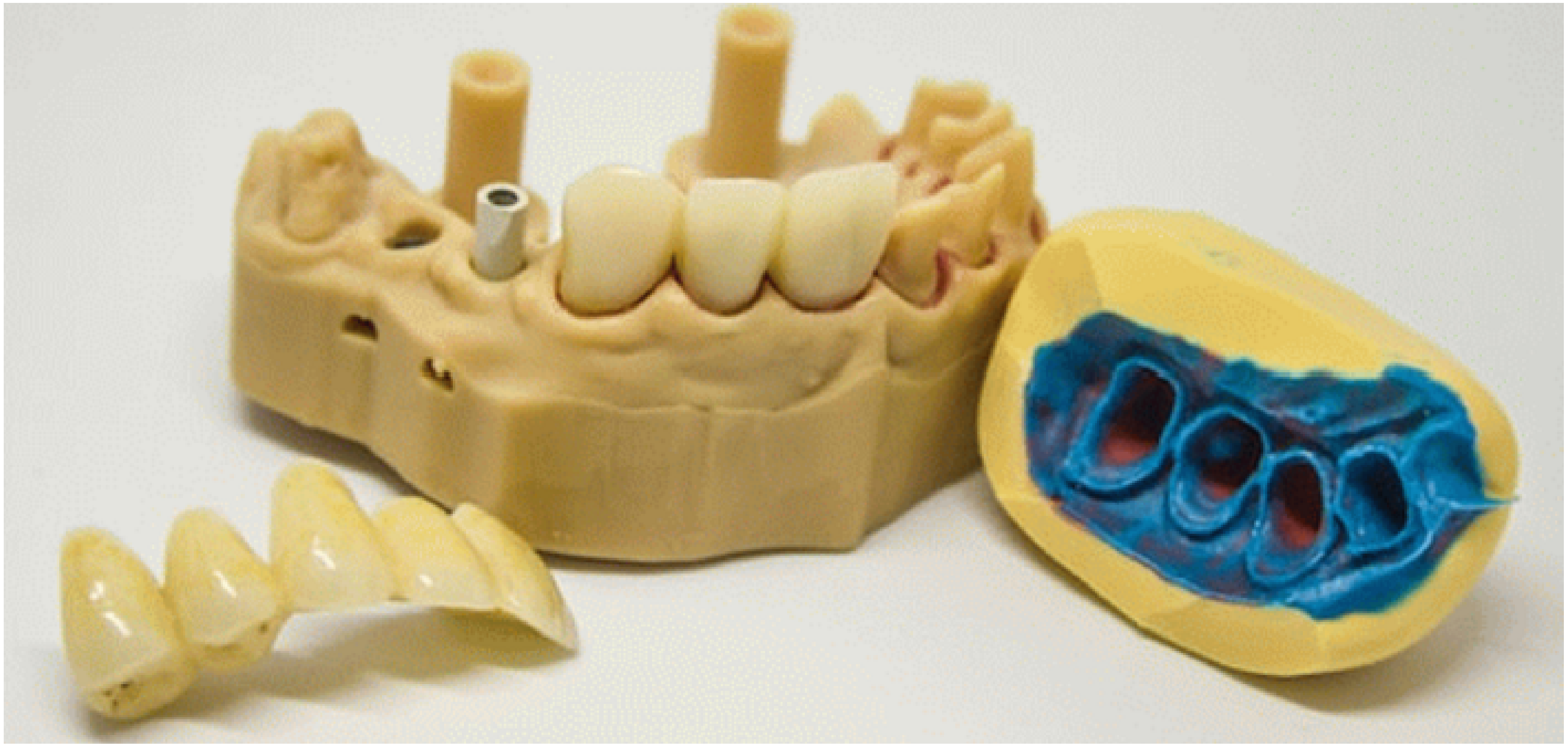
VECTORS in R: Subsetting

Table of Contents

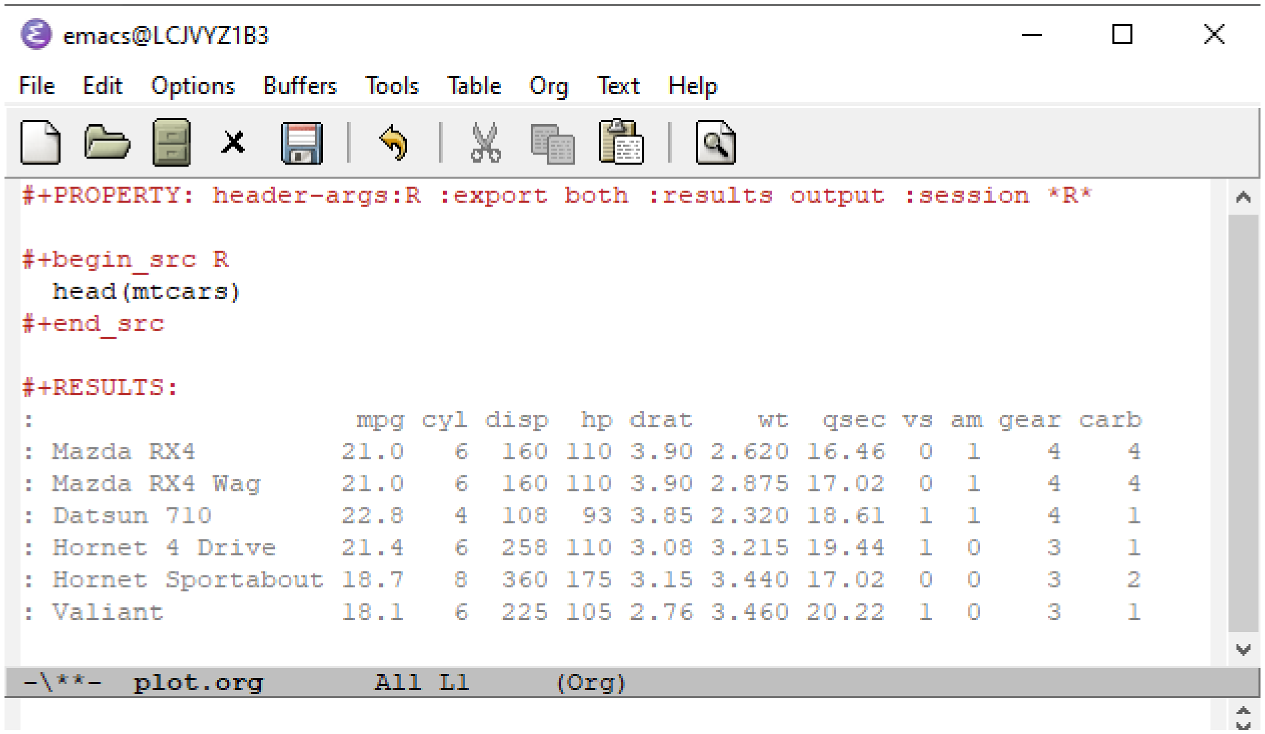


Subsetting and extracting relates to vectorization:

- How to get to parts of a vector
- How to control the indexing
- How to rescale vectors

- How to create matrices and arrays
- How to mix different data types

1 Preparations to code along



```
#+PROPERTY: header-args:R :export both :results output :session *R*

#+begin_src R
  head(mtcars)
#+end_src

#+RESULTS:
:                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
: Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
: Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
: Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
: Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
: Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
: Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

- Open a new Org-mode file `subsetting.org` in Emacs
- Put this line at the top of the file `subsetting.org`

```
#+PROPERTY: header-args:R :results output :session *R*
```

- Activate the code by putting your cursor on the line and entering `C-c C-c`. You should see the message `Local setup has been refreshed` in the minibuffer at the bottom of the editor.
- When you execute your first R code block, you'll be asked where you want the session named `*R*` to run: enter the path to `subsetting.org`
- For plots, use the header `:results graphics file :file subsetting.png` (this will overwrite the PNG file every time you create a new plot)
- When you leave Emacs, you'll be warned that the session `*R*` is active: you can ignore this warning

2 Vectorization 1-2-3

- In R, you don't need loops to perform operations on all elements of a vector - this ability is called *vectorization*.
- There are three forms of vectorization in `R`:

  1. An operator or a function acts on each element of a vector without you having to explicitly write a loop (it's also much faster in terms of execution):

```
foo <- c(1, -1, 4, 4, 0, 59, 3) # sample vector
foo + 3    # add a number to the vector
foo/3.2    # divide vector by number
bar <- foo[-c(4:length(foo))]   # delete part of a vector
```

```
rep(x=bar, times=2)    # repeat a vector
exp(pi*1i) + 1  # Euler's formula - a complex number
class(exp(pi*1i) + 1)
prod(c(1,2,3,4,5))  # 5!
sapply(X=c(5),FUN=factorial) # apply FUN to X
```

2. A function takes a vector as input and calculates a summary statistic:

```
sum(1:5) # sum over all elements
mean(1:5) # average over all elements
summary(1:5) # 5-point statistical summary
```

3. A function calculates a summary statistic from several of its input arguments - this does not always work:

```
sum(1,2,3,4,5)  # OK
mean(1,2) # not OK
mean(c(1,2)) # OK
```

3 Subsetting: retrieving vector elements

We're working with the pre-loaded `Nile` data set.

- Print `Nile`. **What do the numbers at the beginning of each row mean?** (And how can you verify this?)

```
Nile
```

- Never investigate in the dark - always check data structure first:

```
str(Nile)
```

- Subsetting means retrieving a subset of vector elements. For "atomic" vectors (not part of a data frame or list as column vectors), you need to use the `[ ]` index operator.
- Print the first element of `Nile`

```
Nile[1]
```

- What's the corresponding data science (**not R**) question?

» What was the average flow through the Nile in 1871? «

- To extract the time at which a time series was sampled, use `time`, which is a function wrapper of the time series, and also a `ts` object

```r
class(time(Nile))
Nile[1]
time(Nile)[1]
```

- What is the value of the last element of `Nile`?

```r
Nile[length(Nile)]
```

- What's the corresponding question?

  »What was the flow through the river Nile in the last year of observations?«

```r
time(Nile)[length(Nile)]
```

4 Using the colon operator in index

- Create a sample vector `foo`

```r
foo <- c(-1,3.0,4,67,330,-3) # assign vector to foo
foo
```

- You can now use the colon operator : for intervals of indices

```
bar <- foo[2:5]
bar
```

- [ ]

  Check using R: is `foo[n]:foo[m]` the same as `foo[n:m]`?

```
foo # original vector
bar # subset of indices 2:5
baz <- foo[2]:foo[5] # vector built using indices 2 and 5
identical(bar,baz)  # are bar and baz the same?
all.equal(bar,baz)  # are they near equal at last?
```

5 Statistical functions work on subsets

- [ ]

  What is the average (`mean`) of the elements 2 to 5 in `foo`?

```
foo[2:5]
mean(foo[2:5])
mean(foo)
```

- [ ]

  What is the `sum` of the elements 2 to 5 in `foo`?

  ```
  foo
  sum(foo[2:5])
  sum(foo)
  ```

- [ ]

  What is the statistical `summary` of the elements 2 to 5 in `foo`?

  ```
  foo
  summary(foo[2:5])
  summary(foo)
  ```

6 Logical functions in vectors: `<`, `>`, `!=`, `==`

- You can directly use logical operators to subset vectors
- Modify the sample vector `foo`: `-1 3 4 67 330 -3`

1. add a `5` between `4` and `67`
2. add `-99 0 0 44` at the end of the vector

```
foo <- c(-1,3.0,4,67,330,-3) # original vector
foo <- c(foo[1:3],5,foo[4:length(foo)], c(-99,0,0,44))
foo
```

You can also use `append` to append another vector to a vector

```
foo <- c(-1,3.0,4,67,330,-3) # original vector
foo <- c(foo[1:3],5,foo[4:length(foo)])
foo <- append(foo, c(-99,0,0,44))  # alternative method
foo
```

- *Tip: reset your variables with `rm(list=ls())` at any time*

- Create logical sub-vectors of positive and negative elements

```
foo_pos <- c(foo > 0)
foo_pos
foo_neg <- c(foo < 0)
foo_neg
foo_nul <- c(foo == 0)  # what is c(foo=0) ?
foo_nul
```

- What is `c(foo = 0)`?

```
c(foo=0)
```

- What happens if you `sum` the logical index vectors?

```
sum(foo_pos)
sum(foo_neg)
sum(foo_nul)
```

7 Selecting with logical index vectors

- You can now use these subvectors as logical flag or index vectors
- For example, to extract all *positive* elements from `foo`

```
foo[foo_pos]  # using an index vector
foo[foo>0]    # using a logical operator
```

- For example, to extract all *negative* elements from `foo`

```
foo[foo_neg]  # using an index vector
foo[foo<0]    # using a logical operator
```

- For example, to extract all 0 elements from `foo`

```
foo[foo_nul]  # using an index vector
foo[foo==0]    # using a logical operator
```

- Why would you define logical flag vectors instead of using operators?

  Because you can define and alter the index vector definition in ONE place, while you'd have to alter the logical operators in many places in a program.

- What was the flow through the Nile from 1960-1966?

```
t <- time(Nile)
Nile[ t >= 1960 & t <= 1966]
sum(Nile[ t >= 1960 & t <= 1966])
```

8 Negative indices

- The minus operator - removes values with respective indices

- We'll work with our (extended) vector `foo` - you may have to re-run the code block where you first defined it, or re-enter the vector:

  ```
  foo <- c(-1,3.0,4,5,67,330,-3,-99,0,0,44)
  foo
  ```

- [ ]

  Remove the first element of `foo`, then remove the last element of `foo` (without storing), and finally remove both elements simultaneously

  ```
  foo
  foo[-1]
  foo[-length(foo)]
  foo[-c(1,length(foo))]
  ```

- [ ]

  What is the difference between `foo[length(foo)]` and `foo[-length(foo)]`?

  - `foo[(length(foo)]` selects the last element of `foo`
  - `foo[-(length(foo)]` removes the last element of `foo`
- [ ]

I've made an entry mistake: I defined a vector

`vec <- c(5,-2,3,4,6,10,40221,-8)` but I really wanted:

`vec = 5 -2.3 4 6 10 40221 -8` - how can I fix that?

```
vec <- c(5,-2,3,4,6,10,40221,-8)
vec[c(2,3)] # I want to replace these by -2.3
vec[-3]  # delete third element (this will NOT change vec yet)
vec <- vec[-3]  # this will change vec
vec[2] <- -2.3 # overwrite second element (this will change vec)
vec
```

9 Putting dissected vectors back together

- Store the next-to-last value of `vec` in `bar`

```
bar <- vec[length(vec)-1]
bar
```

- Store all other elements of `vec` in `qux`

```
qux <- vec[-(length(vec)-1)]
qux
```

- `[ ]`

Now put `qux` and `bar` together again to get the original `vec` in only one command!

```
c(qux[-length(qux)],bar,qux[length(qux)])
```

1. remove last element of `qux`
2. add `bar` at the end
3. put last element of `qux` back

10 Practice with `Nile`

- Download the raw `6_subsetting_practice.org` [from GitHub](): [tinyurl.com/5fzh98vd]()

- Complete the tasks in class
- When you're done, upload to Canvas

Author: Introduction to data science (DSC 105) Fall 2022

Created: 2022-10-14 Fri 08:38