Working with big data

Programming for Statistical Science

Shawn Santo

Supplementary materials

Full video lecture available in Zoom Cloud Recordings

Additional resources

- Chapter 2, Advanced R by Wickham, H.
- `vroom` vignette

Memory basics

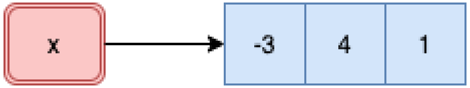
Names and values

In R, a name has a value. It is not the value that has a name.

For example, in

```
x <- c(-3, 4, 1)
```

the object named `x` is a reference to vector `c(-3, 4, 1)`.



We can see where this lives in memory with

```
library(lobstr)
lobstr::obj_addr(x)
```

```
#> [1] "0x7fc4f8fdc048"
```

and its size with

```
lobstr::obj_size(x)
```

```
#> 80 B
```

Copy-on-modify: atomic vectors

Understanding when R creates a copy of an object will allow you to write faster code. This is also important to keep in mind when working with very large vectors.

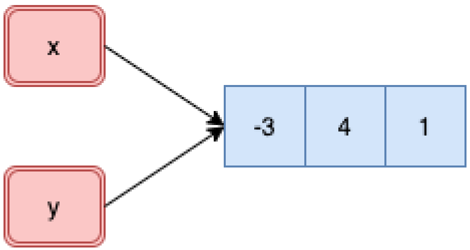
```
x <- c(-3, 4, 1)
y <- x
```

```
obj_addr(x)
```

```
#> [1] "0x7fc4f652a258"
```

```
obj_addr(y)
```

```
#> [1] "0x7fc4f652a258"
```



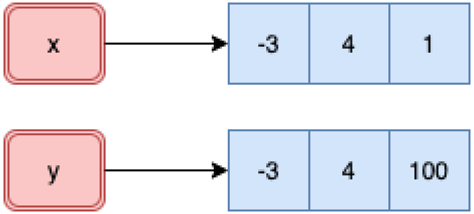
```
y[3] <- 100
```

```
obj_addr(x)
```

```
#> [1] "0x7fc4f652a258"
```

```
obj_addr(y)
```

```
#> [1] "0x7fc4f8060408"
```



```r
x <- c(0, 1, 9)
y <- x

obj_addr(x)
```

```
#> [1] "0x7fc4fca144f8"
```

```r
obj_addr(y)
```

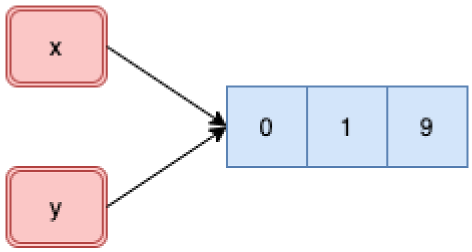
```
#> [1] "0x7fc4fca144f8"
```

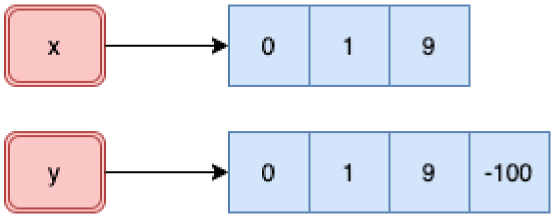
```r
y[4] <- -100
obj_addr(x)
```

```
#> [1] "0x7fc4fca144f8"
```

```r
obj_addr(y)
```

```
#> [1] "0x7fc4f80606d8"
```





Even though only one component changed in the atomic vector `y`, R created a new object as seen by the new address in memory.

Memory tracking

Function `tracemem()` marks an object so that a message is printed whenever the internal code copies the object. Let's see when `x` gets copied.

```
x <- c(0, 1, 1, 2, 3, 5, 8, 13, 21, 34)
tracemem(x)
```

```
#> [1] "<0x7fc4fb90f848>"
```

```
y <- x
```

```
y[1] <- 0
```

```
#> tracemem[0x7fc4fb90f848 -> 0x7fc4f8a01628]: eval eval withVisible withCallingH
```

```
x
```

```
#>  [1]  0  1  1  2  3  5  8 13 21 34
```

```
y
```

```
#>  [1]  0  1  1  2  3  5  8 13 21 34
```

```
c(obj_addr(x), obj_addr(y))
```

```
#> [1] "0x7fc4fb90f848" "0x7fc4f8a01628"
```

```
x[1] <- 0
```

```
#> tracemem[0x7fc4fb90f848 -> 0x7fc4fba6df28]: eval eval withVisible withCallingHa
```

```
lobstr::ref(x)
```

```
#> [1:0x7fc4fba6df28] <dbl>
```

```
lobstr::ref(y)
```

```
#> [1:0x7fc4f8a01628] <dbl>
```

```
untracemem(x)
```

Copy-on-modify: lists

```
x <- list(a = 1, b = 2, c = 3)
obj_addr(x)
```

```
#> [1] "0x7fc4f89f0d98"
```

```
y <- x
```

```
c(obj_addr(x), obj_addr(y))
```

```
#> [1] "0x7fc4f89f0d98" "0x7fc4f89f0d98"
```

```
ref(x, y)
```

```
#> █ [1:0x7fc4f89f0d98] <named list>
#> ├─a = [2:0x7fc4fc824b78] <dbl>
#> ├─b = [3:0x7fc4fc824b40] <dbl>
#> └─c = [4:0x7fc4fc824b08] <dbl>
#>
#> [1:0x7fc4f89f0d98]
```

```r
y$c <- 4
```

```r
ref(x, y)
```

```
#> █ [1:0x7fc4f89f0d98] <named list>
#> ├─a = [2:0x7fc4fc824b78] <dbl>
#> ├─b = [3:0x7fc4fc824b40] <dbl>
#> └─c = [4:0x7fc4fc824b08] <dbl>
#>
#> █ [5:0x7fc4f89f0a78] <named list>
#> ├─a = [2:0x7fc4fc824b78]
#> ├─b = [3:0x7fc4fc824b40]
#> └─c = [6:0x7fc4f8fde508] <dbl>
```

```r
x <- list(a = 1, b = 2, c = 3)
y <- x
```

```r
c(obj_addr(x), obj_addr(y))
```

```
#> [1] "0x7fc4f89f9508" "0x7fc4f89f9508"
```

```r
y$d <- 9
ref(x, y)
```

```
#> █ [1:0x7fc4f89f9508] <named list>
#> ├─a = [2:0x7fc4fc56a728] <dbl>
#> ├─b = [3:0x7fc4fc56a6f0] <dbl>
#> └─c = [4:0x7fc4fc56a6b8] <dbl>
#>
#> █ [5:0x7fc4fd06a738] <named list>
#> ├─a = [2:0x7fc4fc56a728]
#> ├─b = [3:0x7fc4fc56a6f0]
#> ├─c = [4:0x7fc4fc56a6b8]
#> └─d = [6:0x7fc4f8fe8eb0] <dbl>
```

R creates a shallow copy. Shared components exist with elements `a`, `b`, and `c`.

Copy-on-modify: data frames

```
library(tidyverse)
x <- tibble(a = 1:3, b = 9:7)
```

```
ref(x)
```

```
#> █ [1:0x7fc4ff987908] <tibble>
#> ├─a = [2:0x7fc4f862a6b8] <int>
#> └─b = [3:0x7fc4f8605dc8] <int>
```

```
y <- x %>%
  mutate(b = b ^ 2)
```

```
ref(x, y)
```

```
#> █ [1:0x7fc4ff987908] <tibble>
#> ├─a = [2:0x7fc4f862a6b8] <int>
#> └─b = [3:0x7fc4f8605dc8] <int>
#>
#> █ [4:0x7fc4fc727588] <tibble>
#> ├─a = [2:0x7fc4f862a6b8]
#> └─b = [5:0x7fc4fe365738] <dbl>
```

```
z <- x
ref(x, z)
```

```
#> █ [1:0x7fc4ff987908] <tibble>
#> ├─a = [2:0x7fc4f862a6b8] <int>
#> └─b = [3:0x7fc4f8605dc8] <int>
#>
#> [1:0x7fc4ff987908]
```

```
z <- x %>%
  add_row(a = -1, b = -1)
```

```
ref(x, z)
```

```
#> █ [1:0x7fc4ff987908] <tibble>
#> ├─a = [2:0x7fc4f862a6b8] <int>
#> └─b = [3:0x7fc4f8605dc8] <int>
#>
#> █ [4:0x7fc4fc655648] <tibble>
#> ├─a = [5:0x7fc4fbbbb838] <dbl>
#> └─b = [6:0x7fc50001b218] <dbl>
```

If you modify a column, only that column needs to be copied in memory. However, if you modify a row, the entire data frame is copied in memory.

Exercise

Can you diagnose what is going on below?

```
x <- 1:10; y <- x;

tracemem(x)
```

```
#> [1] "<0x7fc4fd056840>"
```

```
c(obj_addr(x), obj_addr(y))
```

```
#> [1] "0x7fc4fd056840" "0x7fc4fd056840"
```

```
y[1] <- 3
```

```
#> tracemem[0x7fc4fd056840 -> 0x7fc4fe2f72f8]: eval eval withVisible withCallingH
#> tracemem[0x7fc4fe2f72f8 -> 0x7fc4fe2b5e98]: eval eval withVisible withCallingH
```

Object size

Object sizes can sometimes be deceiving.

```
x <- rnorm(1e6)
y <- 1:1e6
z <- seq(1, 1e6, by = 1)
s <- (1:1e6) / 2
```

```
c(obj_size(x), obj_size(y), obj_size(z), obj_size(s))
```

```
#> * 8,000,048 B
#> *       680 B
#> * 8,000,048 B
#> * 8,000,048 B
```

```r
c(obj_size(c(1L)), obj_size(c(1.0)))
```

```
#> * 56 B
#> * 56 B
```

```r
c(obj_size(c(1L, 2L)), obj_size(as.numeric(c(1.0, 2.0))))
```

```
#> * 56 B
#> * 64 B
```

```r
c(obj_size(c(1L, 2L, 3L)), obj_size(as.numeric(c(1.0, 2.0, 3.0))))
```

```
#> * 64 B
#> * 80 B
```

```r
c(obj_size(integer(10000)), obj_size(numeric(10000)))
```

```
#> * 40,048 B
#> * 80,048 B
```

There is overhead with creating vectors in R. Take a look at `?Memory` if you want to dig deeper as to the overhead cost.

Exercise

Starting from 0 we can see that

```
lobstr::obj_size(integer(0))
```

```
#> 48 B
```

```
lobstr::obj_size(numeric(0))
```

```
#> 48 B
```

are both 48 bytes. Based on the results on the next slide can you deduce how R handles these numeric data in memory?

```r
diff(sapply(0:100, function(x) lobstr::obj_size(integer(x))))
```

```
#>   [1]  8  0  8  0 16  0  0  0 16  0  0  0 16  0  0  0 64  0  0  0  0  0  0  0
#>  [26]  0  0  0  0  0  0  0  8  0  8  0  8  0  8  0  8  0  8  0  8  0  8  0  8
#>  [51]  8  0  8  0  8  0  8  0  8  0  8  0  8  0  8  0  8  0  8  0  8  0  8  0
#>  [76]  0  8  0  8  0  8  0  8  0  8  0  8  0  8  0  8  0  8  0  8  0  8  0  8
```

```r
c(obj_size(integer(20)), obj_size(integer(22)))
```

```
#> * 176 B
#> * 176 B
```

```r
diff(sapply(0:100, function(x) lobstr::obj_size(numeric(x))))
```

```
#>   [1]  8  8 16  0 16  0 16  0 64  0  0  0  0  0  0  0  8  8  8  8  8  8  8  8
#>  [26]  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8
#>  [51]  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8
#>  [76]  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8
```

```r
c(obj_size(numeric(10)), obj_size(numeric(14)))
```

```
#> * 176 B
#> * 176 B
```

I/O big data

Getting big data into R

Dimensions: 3,185,906 x 9

```
url <- "http://www2.stat.duke.edu/~sms185/data/bike/cbs_2015.csv"
```

```
system.time({x <- read.csv(url)})
```

```
   user  system elapsed
29.739   1.085  37.321
```

```
system.time({x <- readr::read_csv(url)})
```

```
Parsed with column specification:
cols(
  Duration = col_double(),
  `Start date` = col_datetime(format = ""),
  `End date` = col_datetime(format = ""),
  `Start station number` = col_double(),
  `Start station` = col_character(),
  `End station number` = col_double(),
  `End station` = col_character(),
  `Bike number` = col_character(),
  `Member type` = col_character()
)
|=================================| 100%  369 MB
   user  system elapsed
12.773   1.727  22.327
```

```r
system.time({x <- data.table::fread(url)})
```

```
trying URL 'http://www2.stat.duke.edu/~sms185/data/bike/cbs_2015.csv'
Content type 'text/csv' length 387899567 bytes (369.9 MB)
==================================================
downloaded 369.9 MB

   user  system elapsed 
  7.363   2.009  19.942 
```

```r
system.time({x <- vroom::vroom(url)})
```

```
Observations: 3,185,906
Variables: 9
chr  [4]: Start station, End station, Bike number, Member type
dbl  [3]: Duration, Start station number, End station number
dttm [2]: Start date, End date

Call `spec()` for a copy-pastable column specification
Specify the column types with `col_types` to quiet this message

   user  system elapsed 
  5.873   2.361  18.606 
```

Getting bigger data into R

Dimensions: 10,277,677 x 9

```r
url <- "http://www2.stat.duke.edu/~sms185/data/bike/full.csv"
```

```r
system.time({x <- read.csv(url)})
```

```
    user  system elapsed 
119.472   5.037 139.214 
```

```r
system.time({x <- readr::read_csv(url)})
```

```
Parsed with column specification:
cols(
  Duration = col_double(),
  `Start date` = col_datetime(format = ""),
  `End date` = col_datetime(format = ""),
  `Start station number` = col_double(),
  `Start station` = col_character(),
  `End station number` = col_double(),
  `End station` = col_character(),
  `Bike number` = col_character(),
  `Member type` = col_character()
)
|=================================| 100%  1191 MB
   user  system elapsed 
46.845   7.607  87.425 
```

```r
system.time({x <- data.table::fread(url)})
```

```
trying URL 'http://www2.stat.duke.edu/~sms185/data/bike/full.csv'
Content type 'text/csv' length 1249306730 bytes (1191.4 MB)
==================================================
downloaded 1191.4 MB

|--------------------------------------------------|
|==================================================|
   user  system elapsed
33.402   7.249  79.806
```

```r
system.time({x <- vroom::vroom(url)})
```

```
Observations: 10,277,677
Variables: 9
chr  [4]: Start station, End station, Bike number, Member type
dbl  [3]: Duration, Start station number, End station number
dttm [2]: Start date, End date

Call `spec()` for a copy-pastable column specification
Specify the column types with `col_types` to quiet this message
   user  system elapsed
18.837   6.731  57.203
```

Summary

Function	Elapsed Time (s)
`vroom::vroom()`	~57
`data.table::fread()`	~80
`readr::read_csv()`	~87
`read.csv()`	~139

Observations: 10,277,677

Variables: 9

Wrangling big data

Package `dtplyr`

`dtplyr` provides a `data.table` backend for `dplyr`. The goal of `dtplyr` is to allow you to write dplyr code that is automatically translated to the equivalent, but usually much faster, `data.table` code.

```
library(dtplyr)
library(tidyverse)
```

Since it is a backend, you will use `dplyr` verbs (functions) as before.

Get big data

```
base_url <- "https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2019-"
month_ext <- str_pad(1:12, width = 2, pad = "0")
urls <- str_c(base_url, month_ext, ".csv", sep = "")

taxi_2019 <- map_df(urls, vroom)
```

Caution: this full dataset is a dataframe of 84,399,019 x 18.

```
# A tibble: 84,399,019 x 18
   VendorID tpep_pickup_dat… tpep_dropoff_da… passenger_count trip_distance RatecodeID
      <int> <chr>            <chr>                      <int>         <dbl>      <int>
 1        1 2019-01-01 00:4… 2019-01-01 00:5…               1           1.5          1
 2        1 2019-01-01 00:5… 2019-01-01 01:1…               1           2.6          1
 3        2 2018-12-21 13:4… 2018-12-21 13:5…               3           0            1
 4        2 2018-11-28 15:5… 2018-11-28 15:5…               5           0            1
 5        2 2018-11-28 15:5… 2018-11-28 15:5…               5           0            2
 6        2 2018-11-28 16:2… 2018-11-28 16:2…               5           0            1
 7        2 2018-11-28 16:2… 2018-11-28 16:3…               5           0            2
 8        1 2019-01-01 00:2… 2019-01-01 00:2…               1           1.3          1
 9        1 2019-01-01 00:3… 2019-01-01 00:4…               1           3.7          1
10        1 2019-01-01 00:5… 2019-01-01 01:0…               2           2.1          1
# … with 84,399,009 more rows, and 12 more variables: store_and_fwd_flag <chr>,
#   PULocationID <int>, DOLocationID <int>, payment_type <int>, fare_amount <dbl>, extra <dbl>,
#   mta_tax <dbl>, tip_amount <dbl>, tolls_amount <dbl>, improvement_surcharge <dbl>,
#   total_amount <dbl>, congestion_surcharge <dbl>
```

Time comparison

Using dplyr

```
system.time({
 taxi_2019 %>%
   mutate(pickup_datetime  = as_datetime(tpep_pickup_datetime),
          dropoff_datetime = as_datetime(tpep_dropoff_datetime),
          pickup_month     = month(pickup_datetime, label = TRUE),
          pickup_day       = wday(pickup_datetime, label = TRUE)) %>%
   group_by(pickup_month, pickup_day) %>%
   summarise(mean_trip_distance = mean(trip_distance))
})
    user  system elapsed
339.326  21.729 444.383
```

Using dtplyr

```
taxi_2019_lazy <- lazy_dt(taxi_2019)
system.time({
 taxi_2019_lazy %>%
   mutate(pickup_datetime  = as_datetime(tpep_pickup_datetime),
          dropoff_datetime = as_datetime(tpep_dropoff_datetime),
          pickup_month     = month(pickup_datetime, label = TRUE),
          pickup_day       = wday(pickup_datetime, label = TRUE)) %>%
   group_by(pickup_month, pickup_day) %>%
   summarise(mean_trip_distance = mean(trip_distance)) %>%
   as_tibble()
})
    user  system elapsed
384.199  47.111 530.458
```

What's the point of this package?

The benefit comes when

1. you have many many groups (millions);
2. you are sorting;
3. you are doing joins or other merges with large data.

`dtplyr` will always be a little slower than `data.table`. However, this slightly worse performance may be better than learning the sytax of `data.table`.

Going forward

Big data strategies

1. Avoid unnecessary copies of large objects

2. Downsample - you can't exceed $2^31 - 1$ rows, columns, or components

    - Downsample to visualize and use summary statistics
    - Downsample to wrangle and understand
    - Downsample to model

3. Get more RAM - this is not easy or even sometimes an option

4. Parallelize - this is not always an option

    - Execute a chunk and pull strategy

References

1. Data Table Back-End for dplyr. (2020). https://dtplyr.tidyverse.org/index.html.
2. Read and Write Rectangular Text Data Quickly. (2020). https://vroom.r-lib.org/
3. Wickham, H. (2019). Advanced R. https://adv-r.hadley.nz/