Data.Table Package

Table of Contents

Import the cohort and hospital data

We will return to the same cohort and hospital data from before.

```
library(data.table)

## Import Data
cohort <- fread("patient_demo.csv")
hosp <- fread("hosp_demo.csv")
```

Let's also merge in the academic unit variable to the cohort data:

```
## merge in the academic trait
setkey(cohort,hosp_id)
setkey(hosp,hosp_id)
cohort[hosp,academic := academic]
cohort
```

```
##         patient_id hosp_id age gender mortality  SES academic
##      1:     100017       1  70   Male        No  Med      Yes
##      2:     100049       1  64   Male       Yes High      Yes
##      3:     100094       1  71   Male        No  Med      Yes
##      4:     100210       1  72   Male        No  Low      Yes
##      5:     100217       1  75   Male        No  Med      Yes
##     ---
##  99996:     199342     100  71 Female        No  Med       No
##  99997:     199778     100  57   Male        No  Med       No
##  99998:     199925     100  72   Male        No  Low       No
##  99999:     199952     100  82   Male        No High       No
## 100000:     199976     100  60 Female        No  Low       No
```

Long Versus Wide data

- Let's think about the concept of long versus wide data. Let's imagine a hypothetical dataset where we are collecting surveys on patients over time (Repeated measures over time).

- Here is some R Code to create the first dataframe:

```
dt1_wide <-
  data.table('patient_id' = c(100,101,102),
             'survey_baseline' = c(2,5,3),
             'survey_6mos' = c(4,6,4),
             'survey_12mos' = c(3,NA,3))
dt1_wide
```

```
##    patient_id survey_baseline survey_6mos survey_12mos
## 1:        100               2           4            3
## 2:        101               5           6           NA
## 3:        102               3           4            3
```

- If we have the first dataframe, how can we reshape it to get the second dataframe?
- This is called going from 'wide' to 'long'
- There is a function in data.table called `melt()` we can use for this purpose. This originally came from the `reshape2` library, but now the function had been improved/optimized for data.table and implemented in data.table.

```
# go from 'wide' to 'long' by 'melting' the dataframe
dt1_long <-
  melt(dt1_wide,
       id.vars = 'patient_id',
       variable.name = 'survey_time',
       value.name = 'survey_Score')
dt1_long
```

```
##    patient_id     survey_time survey_Score
## 1:        100 survey_baseline            2
## 2:        101 survey_baseline            5
## 3:        102 survey_baseline            3
## 4:        100     survey_6mos            4
## 5:        101     survey_6mos            6
## 6:        102     survey_6mos            4
## 7:        100    survey_12mos            3
```

```
## 8:        101    survey_12mos           NA
## 9:        102    survey_12mos            3
```

```
# remove the missing element
dt1_long <-
  dt1_long[!is.na(survey_Score)]
dt1_long
```

```
##    patient_id     survey_time survey_Score
## 1:        100 survey_baseline            2
## 2:        101 survey_baseline            5
## 3:        102 survey_baseline            3
## 4:        100     survey_6mos            4
## 5:        101     survey_6mos            6
## 6:        102     survey_6mos            4
## 7:        100    survey_12mos            3
## 8:        102    survey_12mos            3
```

- How can we go back to the wide form of the data if we only had the long form?
- this is called going from 'long' to 'wide'
- We use the `dcast()` function (also within data.table)

```
# go from long to wide
# notice the missing data element was inferred!
dcast(dt1_long,
      formula = patient_id ~survey_time,
      value.var = 'survey_Score')
```

```
##    patient_id survey_baseline survey_6mos survey_12mos
## 1:        100               2           4            3
## 2:        101               5           6           NA
## 3:        102               3           4            3
```

- Remember - it is often useful and worth considering - should I melt my data down into a long format to make work easier?

More melting / casting data long to wide

We can generalize this beyond just measurements over time and reshaping. Once we have data in a 'long format', it is often easy to do various group by summary calculations, then 'cast' the data up to the format we want it in.

```r
## Calculate overall mortality rate
cohort[,list(mortality_rate=sum(mortality=='Yes')/.N)]
```

```
##    mortality_rate
## 1:        0.17989
```

```r
## Calculate mortality rate by hospital and gender
mortality_Rate_by_hosp <-
  cohort[,list(mortality_rate=sum(mortality=='Yes')/.N),
         by=list(gender,hosp_id)]
mortality_Rate_by_hosp
```

```
##      gender hosp_id mortality_rate
##   1:   Male       1      0.1874299
##   2: Female       1      0.1262136
##   3:   Male       2      0.1820303
##   4: Female       2      0.1770833
##   5: Female       3      0.1397849
##  ---
## 196: Female      98      0.1504425
## 197:   Male      99      0.2057206
## 198: Female      99      0.1682243
## 199:   Male     100      0.1806854
## 200: Female     100      0.1782178
```

```r
## dcast to prettier format
dcast(mortality_Rate_by_hosp,hosp_id ~ gender)
```

```
## Using 'mortality_rate' as value column. Use 'value.var' to override
```

```
##      hosp_id     Female      Male
##   1:       1 0.12621359 0.1874299
##   2:       2 0.17708333 0.1820303
##   3:       3 0.13978495 0.1816118
##   4:       4 0.21978022 0.2114094
##   5:       5 0.11627907 0.1993166
##   6:       6 0.11494253 0.2235551
##   7:       7 0.14814815 0.1647856
##   8:       8 0.20652174 0.2105832
##   9:       9 0.15306122 0.1870101
##  10:      10 0.13131313 0.1761298
##  11:      11 0.09259259 0.1940463
##  12:      12 0.16470588 0.1598272
##  13:      13 0.13483146 0.1820276
##  14:      14 0.15730337 0.1979522
```

```
##  15:      15 0.14432990 0.1743017
##  16:      16 0.17037037 0.1779207
##  17:      17 0.11111111 0.1735967
##  18:      18 0.11458333 0.1644295
##  19:      19 0.07228916 0.1722282
##  20:      20 0.12037037 0.1723426
##  21:      21 0.16842105 0.1618943
##  22:      22 0.17204301 0.1585233
##  23:      23 0.22680412 0.1856823
##  24:      24 0.11504425 0.1570796
##  25:      25 0.17475728 0.1783724
##  26:      26 0.09259259 0.1764032
##  27:      27 0.11111111 0.1762712
##  28:      28 0.05813953 0.1675504
##  29:      29 0.10309278 0.2111597
##  30:      30 0.11363636 0.1529680
##  31:      31 0.25000000 0.1864989
##  32:      32 0.15841584 0.1776815
##  33:      33 0.10679612 0.1566265
##  34:      34 0.16190476 0.1857451
##  35:      35 0.16483516 0.1896930
##  36:      36 0.13636364 0.1894852
##  37:      37 0.18823529 0.1735722
##  38:      38 0.13559322 0.1816168
##  39:      39 0.18085106 0.1729730
##  40:      40 0.15463918 0.1947308
##  41:      41 0.18750000 0.1746575
##  42:      42 0.17045455 0.1740139
##  43:      43 0.15841584 0.1798483
##  44:      44 0.14141414 0.1853998
##  45:      45 0.22330097 0.1706485
##  46:      46 0.10112360 0.1805556
##  47:      47 0.11881188 0.1673961
##  48:      48 0.17475728 0.1911765
##  49:      49 0.13414634 0.1765339
##  50:      50 0.16666667 0.1995565
##  51:      51 0.17475728 0.1877095
##  52:      52 0.21296296 0.1666667
##  53:      53 0.10891089 0.1786108
##  54:      54 0.12380952 0.1676436
##  55:      55 0.16666667 0.1853107
##  56:      56 0.19607843 0.2044693
##  57:      57 0.17977528 0.1932059
```

```
##  58:      58 0.10989011 0.1756440
##  59:      59 0.18691589 0.2029756
##  60:      60 0.14678899 0.1762115
##  61:      61 0.15151515 0.1726776
##  62:      62 0.12745098 0.1745335
##  63:      63 0.19587629 0.1872146
##  64:      64 0.10434783 0.2074866
##  65:      65 0.17757009 0.2118644
##  66:      66 0.19047619 0.1704180
##  67:      67 0.17741935 0.1980519
##  68:      68 0.15217391 0.1907824
##  69:      69 0.14678899 0.1854664
##  70:      70 0.13402062 0.2123288
##  71:      71 0.24324324 0.1856639
##  72:      72 0.13761468 0.1864407
##  73:      73 0.22727273 0.1923937
##  74:      74 0.16666667 0.2041049
##  75:      75 0.18461538 0.1590146
##  76:      76 0.20588235 0.1944444
##  77:      77 0.10833333 0.1674208
##  78:      78 0.18750000 0.2050663
##  79:      79 0.19587629 0.1913043
##  80:      80 0.25000000 0.1744186
##  81:      81 0.23404255 0.1825994
##  82:      82 0.20000000 0.2021277
##  83:      83 0.15384615 0.1750547
##  84:      84 0.06122449 0.2062212
##  85:      85 0.11111111 0.1781946
##  86:      86 0.18518519 0.1796009
##  87:      87 0.13402062 0.1810748
##  88:      88 0.18095238 0.1620763
##  89:      89 0.10784314 0.1939462
##  90:      90 0.14893617 0.1649832
##  91:      91 0.15740741 0.1909846
##  92:      92 0.13402062 0.1844978
##  93:      93 0.11904762 0.1764057
##  94:      94 0.14285714 0.1581292
##  95:      95 0.16666667 0.1586592
##  96:      96 0.17475728 0.1883408
##  97:      97 0.11650485 0.1790541
##  98:      98 0.15044248 0.1842105
##  99:      99 0.16822430 0.2057206
```

```
## 100:     100 0.17821782 0.1806854
##      hosp_id     Female      Male
```

Tables and memory management

Another consideration to remember is actively managing your memory as you create intermediate data.table objects. I often end up creating lookup tables, or other data summaries / transitions that are only briefly required in the middle of my analyses.

Remember, you can always remove a table using `rm()`, and you can check how many tables and what their memory footprint is using `tables()`

```
## remove the hospital data
rm(hosp)
## check tables
tables()
```

```
##                       NAME    NROW NCOL MB
## 1:                  cohort 100,000    7  4
## 2:               dt1_long       8    3  0
## 3:               dt1_wide       3    4  0
## 4: mortality_Rate_by_hosp     200    3  0
##                                                      COLS     KEY
## 1:     patient_id,hosp_id,age,gender,mortality,SES,... hosp_id
## 2:                  patient_id,survey_time,survey_Score
## 3: patient_id,survey_baseline,survey_6mos,survey_12mos
## 4:                        gender,hosp_id,mortality_rate
## Total: 4MB
```