data.table

UseR Bochum #4

Dipl. Soz.Wiss Sebastian Jeworutzki

22.11.2017

Das Paket

- Data.table ist ein schneller Ersatz für data.frames
- Das Paket bringt u.a. Funktionen zum einlesen, speichern, aggregieren und reshapen mit

data.table | CSV Daten einlesen

```r
library(data.table)

# Base R
system.time(DF1 <-read.csv("test.csv", stringsAsFactors=FALSE))
```

```
   user  system elapsed
 20.202   0.180  20.828
```

```r
# data.table
system.time(DT <- fread("test.csv", stringsAsFactors=FALSE))
```

```
   user  system elapsed
  1.466   0.071   3.027
```

data.table | Ein Beispiel

```
dt <- fread("https://raw.githubusercontent.com/wiki/arunsrinivasan/
  ↪   flights/NYCflights14/flights14.csv")
dt
```

```
        year month day dep_time dep_delay arr_time arr_delay
          ↪   cancelled
     1: 2014     1   1      914        14     1238        13
       ↪   0
     2: 2014     1   1     1157        -3     1523        13
       ↪   0
     3: 2014     1   1     1902         2     2224         9
       ↪   0
     4: 2014     1   1      722        -8     1014       -26
       ↪   0
     5: 2014     1   1     1347         2     1706         1
       ↪   0
    ---
253312: 2014    10  31     1459         1     1747       -30
  ↪   0
253313: 2014    10  31      854        -5     1147       -14
  ↪   0
253314: 2014    10  31     1102        -8     1311        16
  ↪   0
253315: 2014    10  31     1106        -4     1325        15
  ↪   0
```

data.table | der []-Operator

  - Variablennamen können ohne **$** und " " genutzt werden
- Funktionsaufrufe können direkt auf Spalten angewandt werden

```
dt[year==2014, arr_delay]       # Vektor

dt[year==2014, "arr_delay"]     # DT erzwingen (entspricht drop=F)

dt[year==2014, mean(arr_delay)] # Funktionsaufruf
```

data.table | der []-Operator

- Mehre Spalten werden nicht über einen Vektor `c()` sondern eine Liste `list()` ausgewählt
- Die Kurzform für eine Liste ist `.()`

```
dt[origin=="JFK", .(year, arr_delay)]
```

```
        year arr_delay
     1: 2014        13
     2: 2014        13
     3: 2014         9
     4: 2014         1
     5: 2014       -18
    ---
 81479: 2014       -21
 81480: 2014       -37
 81481: 2014       -33
 81482: 2014       -38
 81483: 2014       -38
```

data.table | der []-Operator

dt[which row?, what to do?, grouped by?]

data.table | der []-Operator

- Aggregation über eine oder mehrere Gruppierungsvariable ist einfach, wenn Gruppierungsvariablen angegeben sind
- Der Operator `.N` gibt die Anzahl der Zeilen pro Gruppe an

```
bymonth <- dt[, .(MeanArrDelay=mean(arr_delay),
                  nFlights=.N), .(month, origin)]
bymonth
```

```
    month origin MeanArrDelay nFlights
 1:     1    JFK   24.0154821     7105
 2:     1    LGA   16.2798239     7494
 3:     1    EWR   22.0744175     8197
 4:     2    JFK   21.1864582     6779
 5:     2    LGA   12.0515116     6814
---
26:     9    LGA   -0.6750238     8416
27:     9    EWR    2.3048210     8546
28:    10    EWR    0.3053586     8603
29:    10    LGA    3.0418789     8835
30:    10    JFK    1.8921557     8605
```

data.table | sortieren

```
setorder(bymonth, origin)
bymonth
```

```
    month origin MeanArrDelay nFlights
 1:     1    EWR   22.0744175     8197
 2:     2    EWR   19.0084488     7220
 3:     3    EWR    5.8279173     9478
 4:     4    EWR   14.8373445     8601
 5:     5    EWR    9.8646334     8525
---
26:     6    LGA    8.5373408     8717
27:     7    LGA    9.8095184     8825
28:     8    LGA    3.7797586     9031
29:     9    LGA   -0.6750238     8416
30:    10    LGA    3.0418789     8835
```

data.table | reshape

Das reshapen mit `data.table` funktioniert mit den Befehlen `dcast` und `melt` (vergleichbar aus dem `reshape2`-Paket ...

```
wide <- dcast(bymonth, origin ~  month,
              value.var=c("MeanArrDelay", "nFlights"))
wide
```

```
   origin MeanArrDelay_1 MeanArrDelay_2 MeanArrDelay_3 MeanArrDelay_4
1:    EWR       22.07442       19.00845       5.827917      14.837344
2:    JFK       24.01548       21.18646       3.090391       3.785626
3:    LGA       16.27982       12.05151       4.636562       3.083436
   MeanArrDelay_5 MeanArrDelay_6 MeanArrDelay_7 MeanArrDelay_8
1:       9.864633      11.775698      12.894581       3.912435
2:       5.448886       5.839349      14.146392       3.053750
3:       7.960193       8.537341       9.809518       3.779759
   MeanArrDelay_9 MeanArrDelay_10 nFlights_1 nFlights_2 nFlights_3
1:      2.3048210       0.3053586       8197       7220       9478
2:     -0.2403986       1.8921557       7105       6779       8253
3:     -0.6750238       3.0418789       7494       6814       8692
   nFlights_4 nFlights_5 nFlights_6 nFlights_7 nFlights_8 nFlights_9
1:       8601       8525       9349       9448       9433       8546
2:       8070       8305       8422       8730       8986       8228
3:       8917       8692       8717       8825       9031       8416
   nFlights_10
1:        8603
```

data.table | reshape

... die Funktionen arbeiten aber viel schneller.

```
long<- melt(wide,
            id="origin",
            measure=patterns("MeanArrDelay_", "nFlights_"))
long
```

```
    origin variable     value1 value2
 1:    EWR        1 22.0744175   8197
 2:    JFK        1 24.0154821   7105
 3:    LGA        1 16.2798239   7494
 4:    EWR        2 19.0084488   7220
 5:    JFK        2 21.1864582   6779
---
26:    JFK        9 -0.2403986   8228
27:    LGA        9 -0.6750238   8416
28:    EWR       10  0.3053586   8603
29:    JFK       10  1.8921557   8605
30:    LGA       10  3.0418789   8835
```

data.table

- Mit `data.table` sind noch viel komplexere Datenaufbereitungen möglich
- Ein guter Startpunkt für weitere Tipps ist r-datatable.com