Data.Table Package

Table of Contents

Base R

- In base R, we use the 'dataframe' to organize data.
- We also use `read.csv()` to import data. This can be slow though…
- You may also be familiar with the `tidyverse`, or specifically `dplyr` for managing data in R.
- Why is base R slow?
  - Bad r code (not pre-allocating result vectors)
  - Lazy copying when manipulating dataframes (like renaming)
  - Slower sort algorithms
  - Not leveraging sort order

The Data.Table Package

- A different approach to dataframes in R
- data.table replaces the data.frame
- A data.table object is also a data.frame
  - You can use a data.table anywhere you can use a data.frame
- Goals of the data.table package
  - Reduce time to write code through sensical syntax (opinion)
  - Reduce the time code takes to run through efficient copying ordered grouping and ordered joins (fact)

- – Some of the computing time gains depend on sorting (setting keys) the data.table then leveraging the sort order for joins, group by calculations, and subsetting

Creating a data.table

- We can directly create a data.table similar to a data.frame
- We can easily convert a data.frame to a data.table
- We can use `fread()` to read in structured text as a data.table, this replaces`read.csv()`

```
library(data.table)

## We can directly create a data.table just like a dataframe
dt1 <-
  data.table(x1=1:10,
             x2=10:1)
dt1
```

```
##     x1 x2
##  1:  1 10
##  2:  2  9
##  3:  3  8
##  4:  4  7
##  5:  5  6
##  6:  6  5
##  7:  7  4
##  8:  8  3
##  9:  9  2
## 10: 10  1
```

```
str(dt1)
```

```
## Classes 'data.table' and 'data.frame':   10 obs. of  2 variables:
##  $ x1: int  1 2 3 4 5 6 7 8 9 10
##  $ x2: int  10 9 8 7 6 5 4 3 2 1
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
is.data.frame(dt1)
```

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

```r
is.data.table(dt1)

## [1] TRUE
```

Creating a data.table 2

- We can easily convert a data.frame to a data.table
- We can use `fread` to read in structured text as a data.table (this replaces `read.csv`)

```r
library(data.table)
library(plyr) # for the baseball data

## We can convert existing dataframe to dt
data(baseball)
str(baseball)

## 'data.frame':    21699 obs. of  22 variables:
##  $ id   : chr  "ansonca01" "forceda01" "mathebo01" "startjo01" ...
##  $ year : int  1871 1871 1871 1871 1871 1871 1871 1872 1872
1872 ...
##  $ stint: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ team : chr  "RC1" "WS3" "FW1" "NY2" ...
##  $ lg   : chr  "" "" "" "" ...
##  $ g    : int  25 32 19 33 29 29 29 46 37 25 ...
##  $ ab   : int  120 162 89 161 128 146 145 217 174 130 ...
##  $ r    : int  29 45 15 35 35 40 36 60 26 40 ...
##  $ h    : int  39 45 24 58 45 47 37 90 46 53 ...
##  $ X2b  : int  11 9 3 5 3 6 5 10 3 11 ...
##  $ X3b  : int  3 4 1 1 7 5 7 7 0 0 ...
##  $ hr   : int  0 0 0 1 3 1 2 0 0 0 ...
##  $ rbi  : int  16 29 10 34 23 21 23 50 15 16 ...
##  $ sb   : int  6 8 2 4 3 2 2 6 0 2 ...
##  $ cs   : int  2 0 1 2 1 2 2 6 1 2 ...
##  $ bb   : int  2 4 2 3 1 4 9 16 1 1 ...
##  $ so   : int  1 0 0 0 0 1 1 3 1 0 ...
##  $ ibb  : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ hbp  : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ sh   : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ sf   : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ gidp : int  NA NA NA NA NA NA NA NA NA NA ...

baseball_dt <-
  data.table(baseball)
```

```
## Fast File Read
cohort <- fread("patient_demo.csv")

## how much faster?
system.time(cohort <- read.csv("patient_demo.csv"))

##    user  system elapsed
##   0.147   0.012   0.161

library(readr)
system.time(cohort <- read_csv("patient_demo.csv"))

##
## ── Column specification
────────────────────────────────────────────────────────

## cols(
##   patient_id = col_double(),
##   hosp_id = col_double(),
##   age = col_double(),
##   gender = col_character(),
##   mortality = col_character(),
##   SES = col_character()
## )

##    user  system elapsed
##   0.226   0.004   0.334

system.time(cohort <- fread("patient_demo.csv"))

##    user  system elapsed
##   0.038   0.000   0.023
```

data.table Commands

- Recall a data.table is also a data.frame so all the data.frame methods work
- Data.table prints only the first 5 and last 5 rows of a table if the table is larger than 100 rows

```
## list all data.tables and their size
tables()

##             NAME    NROW NCOL MB
COLS KEY
## 1: baseball_dt  21,699   22  2
id,year,stint,team,lg,g,...
```

```
## 2:      cohort 100,000    6  3
patient_id,hosp_id,age,gender,mortality,SES
## 3:         dt1      10    2  0
x1,x2
## Total: 5MB
```

```
## Confirm this is a data.table (and a data frame)
class(cohort)
```

```
## [1] "data.table" "data.frame"
```

```
## the usual methods for data.frames all work for
summary(cohort)
```

```
##    patient_id        hosp_id            age             gender

##  Min.   :100000   Min.   :  1.00   Min.   : 22.00   Length:100000

##  1st Qu.:125000   1st Qu.: 25.00   1st Qu.: 58.00
Class :character
##  Median :150000   Median : 51.00   Median : 65.00
Mode  :character
##  Mean   :150000   Mean   : 50.55   Mean   : 64.93

##  3rd Qu.:174999   3rd Qu.: 75.00   3rd Qu.: 72.00

##  Max.   :199999   Max.   :100.00   Max.   :109.00

##   mortality              SES
##  Length:100000      Length:100000
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
## Data.tables print the first and last 5 rows
cohort
```

```
##         patient_id hosp_id age gender mortality  SES
##      1:     100000      77  70   Male       Yes  Med
##      2:     100001      45  64   Male        No  Low
##      3:     100002       5  76   Male        No  Med
##      4:     100003      13  51   Male        No High
##       5:     100004      70  76   Male         No  Low
##     ---                                             
##   99996:     199995      60  66   Male         No  Med
##   99997:     199996      14  71   Male         No  Low
##   99998:     199997      39  73   Male        Yes  Med
##   99999:     199998      21  60 Female         No High
## 100000:     199999      81  61   Male         No  Low
```

The i= Argument

- We will use the square brackets to interact with data.tables (the subset operator)

- Forget everything you know about the square brackets (sorry!), it is different in data.table

- The first argument to the square brackets is `i=`, which we use to subset the rows of the data.table

- This can be an integer vector or a Boolean vector

- Any references to variables will be first resolved within the data.table environment

```
cohort[] # prints all

##         patient_id hosp_id age gender mortality  SES
##       1:     100000      77  70   Male        Yes  Med
##       2:     100001      45  64   Male         No  Low
##       3:     100002       5  76   Male         No  Med
##       4:     100003      13  51   Male         No High
##       5:     100004      70  76   Male         No  Low
##     ---                                             
##   99996:     199995      60  66   Male         No  Med
##   99997:     199996      14  71   Male         No  Low
##   99998:     199997      39  73   Male        Yes  Med
##   99999:     199998      21  60 Female         No High
## 100000:     199999      81  61   Male         No  Low

## the first row
cohort[i=1]
```

```
##    patient_id hosp_id age gender mortality SES
## 1:     100000      77  70   Male       Yes Med

## we usually don't bother naming the argument with i=
## Rows 10 to 20
cohort[10:20]

##     patient_id hosp_id age gender mortality  SES
##  1:     100009      64  71   Male        No  Low
##  2:     100010      43  83   Male        No  Low
##  3:     100011      62  66   Male        No  Low
##  4:     100012      54  58   Male        No  Med
##  5:     100013      69  82   Male        No  Low
##  6:     100014      20  72   Male        No  Med
##  7:     100015      46  48   Male        No High
##  8:     100016      19  71   Male        No High
##  9:     100017       1  70   Male        No  Med
## 10:     100018      81  60   Male        No  Med
## 11:     100019       5  76   Male        No  Med
```

The i= Argument

```
## all patients older than 50
cohort[age>50]

##        patient_id hosp_id age gender mortality  SES
##     1:     100000      77  70   Male       Yes  Med
##     2:     100001      45  64   Male        No  Low
##     3:     100002       5  76   Male        No  Med
##     4:     100003      13  51   Male        No High
##     5:     100004      70  76   Male        No  Low
##    ---
## 92508:     199995      60  66   Male        No  Med
## 92509:     199996      14  71   Male        No  Low
## 92510:     199997      39  73   Male       Yes  Med
## 92511:     199998      21  60 Female        No High
## 92512:     199999      81  61   Male        No  Low

## all patients older than 50 and male
cohort[age>50 & gender == "Male"]

##        patient_id hosp_id age gender mortality  SES
##     1:     100000      77  70   Male       Yes  Med
##     2:     100001      45  64   Male        No  Low
##     3:     100002       5  76   Male        No  Med
##     4:     100003      13  51   Male        No High
##     5:     100004      70  76   Male        No  Low
##    ---
## 83291:     199994      65  85   Male        No  Med
## 83292:     199995      60  66   Male        No  Med
## 83293:     199996      14  71   Male        No  Low
## 83294:     199997      39  73   Male       Yes  Med
## 83295:     199999      81  61   Male        No  Low
```

Question: Select the first 10 rows of the baseball data
Question: Select the rows after the year 2000 for the baseball data
Question: Select the rows with team equal to CLE or NYN

- Answer

```
baseball_dt[1:10]

##             id year stint team lg  g  ab  r  h X2b X3b hr rbi sb cs
bb so ibb
##  1: ansonca01 1871     1  RC1    25 120 29 39  11   3  0  16  6  2
2  1  NA
##  2: forceda01 1871     1  WS3    32 162 45 45   9   4  0  29  8  0
4  0  NA
##  3: mathebo01 1871     1  FW1    19  89 15 24   3   1  0  10  2  1
2  0  NA
##  4: startjo01 1871     1  NY2    33 161 35 58   5   1  1  34  4  2
3  0  NA
##  5: suttoez01 1871     1  CL1    29 128 35 45   3   7  3  23  3  1
1  0  NA
##  6: whitede01 1871     1  CL1    29 146 40 47   6   5  1  21  2  2
4  1  NA
##  7:  yorkto01 1871     1  TRO    29 145 36 37   5   7  2  23  2  2
9  1  NA
##  8: ansonca01 1872     1  PH1    46 217 60 90  10   7  0  50  6  6
16  3  NA
##  9: burdoja01 1872     1  BR2    37 174 26 46   3   0  0  15  0  1
1  1  NA
## 10: forceda01 1872     1  TRO    25 130 40 53  11   0  0  16  2  2
1  0  NA
##     hbp sh sf gidp
##  1:  NA NA NA   NA
##  2:  NA NA NA   NA
##  3:  NA NA NA   NA
##  4:  NA NA NA   NA
##  5:  NA NA NA   NA
##  6:  NA NA NA   NA
##  7:  NA NA NA   NA
##  8:  NA NA NA   NA
##  9:  NA NA NA   NA
## 10:  NA NA NA   NA
```

```
baseball_dt[year > 2000]
```

```
##              id year stint team lg   g  ab   r   h X2b X3b hr rbi
sb cs bb so
##    1: alomaro01 2001     1  CLE AL 157 575 113 193  34  12 20 100
30  6 80 71
##    2: alomasa02 2001     1  CHA AL  70 220  17  54   8   1  4  21
1  2 12 17
##    3: anderbr01 2001     1  BAL AL 131 430  50  87  12   3  8  45
12  4 60 77
##    4: baineha01 2001     1  CHA AL  32  84   3  11   1   0  0   6
0  0  8 16
##    5:  bellda01 2001     1  SEA AL 135 470  62 122  28   0 15  64
2  1 28 59
##   ---

## 1266: benitar01 2007     2  FLO NL  34   0   0   0   0   0  0   0
0  0  0  0
## 1267: benitar01 2007     1  SFN NL  19   0   0   0   0   0  0   0
0  0  0  0
## 1268: ausmubr01 2007     1  HOU NL 117 349  38  82  16   3  3  25
6  1 37 74
## 1269:  aloumo01 2007     1  NYN NL  87 328  51 112  19   1 13  49
3  0 27 30
## 1270: alomasa02 2007     1  NYN NL   8  22   1   3   1   0  0   0
0  0  0  3
##       ibb hbp sh sf gidp
##    1:   5   4  9  9    9
##    2:   1   2  3  2    6
##    3:   4   8  2  1    3
##    4:   0   0  0  2    2
##    5:   1   3  5  4    8
##   ---
## 1266:   0   0  0  0    0
## 1267:   0   0  0  0    0
## 1268:   3   6  4  1   11
## 1269:   5   2  0  3    13
## 1270:   0   0  0  0     0
```

```
baseball_dt[team %in% c("NYN","CLE")]
```

```
##                id year stint team lg   g  ab  r   h X2b X3b hr rbi sb
cs bb  so
##    1: hallmbi01 1901     1  CLE AL   5  19  2   4   0   0  0   3  0
NA  2  NA
##    2: mooreea01 1901     1  CLE AL  31  99  5  16   0   0  0   6  1
NA  6  NA
##    3: weyhigu01 1901     1  CLE AL   2   5  0   0   0   0  0   0  0
NA  0  NA
##    4: hickmch01 1902     2  CLE AL 102 426 61 161  31  11  8  94  8
NA 12  NA
##    5: lajoina01 1902     2  CLE AL  86 348 81 132  35   5  7  64 19
NA 19  NA
##   ---

## 1461: easleda01 2007     1  NYN NL  76 193 24  54   6   0 10  26  0
1 19  35
## 1462: delgaca01 2007     1  NYN NL 139 538 71 139  30   0 24  87  4
0 52 118
## 1463: coninje01 2007     2  NYN NL  21  41  2   8   2   0  0   5  0
0  7   8
## 1464:  aloumo01 2007     1  NYN NL  87 328 51 112  19   1 13  49  3
0 27  30
## 1465: alomasa02 2007     1  NYN NL   8  22  1   3   1   0  0   0  0
0  0   3
##       ibb hbp sh sf gidp
##    1:  NA   0  0 NA   NA
##    2:  NA   0  3 NA   NA
##    3:  NA   0  0 NA   NA
##    4:  NA   3  8 NA   NA
##    5:  NA   6  8 NA   NA
##   ---
## 1461:   1   5  0  1     2
## 1462:   8  11  0  6    12
## 1463:   2   0  1  1     1
## 1464:   5   2  0  3    13
## 1465:   0   0  0  0     0
```

Data.Table: the “j” argument

- The second argument to the square brackets is the `j` argument
- We leave the first argument blank (the `i` argument) if we want all the rows and just put a comma
- The `j` argument can be any valid R expression
- If the `j` argument resolves to a list, a data.table is returned

```
## Returns age as a vector
cohort[,j=age][1:10]

##  [1] 70 64 76 51 76 60 55 66 75 71

## we typically omit the j= piece
cohort[,age][1:10]

##  [1] 70 64 76 51 76 60 55 66 75 71

## Adding list() returns a data.table
cohort[,list(age)]

##         age
##      1:  70
##      2:  64
##      3:  76
##      4:  51
##      5:  76
##     ---
##  99996:  66
##  99997:  71
##  99998:  73
##  99999:  60
## 100000:  61

cohort[,list(age,patient_id)]

##         age patient_id
##      1:  70     100000
##      2:  64     100001
##      3:  76     100002
##      4:  51     100003
##      5:  76     100004
##     ---
##  99996:  66     199995
##  99997:  71     199996
##  99998:  73     199997
##  99999:  60     199998
## 100000:  61     199999
```

```
## use . as a shortcut for list here
cohort[,.(age,patient_id)]
```

```
##         age patient_id
##      1:  70     100000
##      2:  64     100001
##      3:  76     100002
##      4:  51     100003
##      5:  76     100004
##     ---
##  99996:  66     199995
##  99997:  71     199996
##  99998:  73     199997
##  99999:  60     199998
## 100000:  61     199999
```

Data.Table: the “j” argument

Question: From the Baseball data, return a data.table with the rows from after the year 2000 with only the id, year and team columns using the `list()` command
Question: Repeat this using the `.` to replace the `list` command
Question: How many rows meet this condition?

- Answer

```
baseball_dt[year > 2000,list(id,year,team)]
```

```
##             id year team
##    1: alomaro01 2001  CLE
##    2: alomasa02 2001  CHA
##    3: anderbr01 2001  BAL
##    4: baineha01 2001  CHA
##    5:  bellda01 2001  SEA
##   ---
## 1266: benitar01 2007  FLO
## 1267: benitar01 2007  SFN
## 1268: ausmubr01 2007  HOU
## 1269:  aloumo01 2007  NYN
## 1270: alomasa02 2007  NYN
```

```
baseball_dt[year > 2000,.(id,year,team)]
```

```
##               id year team
##    1: alomaro01 2001  CLE
##    2: alomasa02 2001  CHA
##    3: anderbr01 2001  BAL
##    4: baineha01 2001  CHA
##    5:  bellda01 2001  SEA
##   ---
## 1266: benitar01 2007  FLO
## 1267: benitar01 2007  SFN
## 1268: ausmubr01 2007  HOU
## 1269:  aloumo01 2007  NYN
## 1270: alomasa02 2007  NYN
```

```
## How many rows?
nrow(baseball_dt[year > 2000,.(id,year,team)])
```

```
## [1] 1270
```

Using Integers and Characters in the j argument

- In base R we can use integers or characters to reference columns as the second argument following `[`

- The same behavior is true in data.table (this was not always the case in prior versions of data.table)

- However, one difference is the class of the return object. Notice data.table returns a data.table, while the data.frame would return a vector instead of a dataframe.

```
# These work as expected
cohort[,1]
```

```
##        patient_id
##      1:     100000
##      2:     100001
##      3:     100002
##      4:     100003
##      5:     100004
##     ---
##  99996:     199995
##  99997:     199996
##  99998:     199997
##  99999:     199998
## 100000:     199999
```

```
cohort[,"age"]
```

```
##         age
##      1:  70
##      2:  64
##      3:  76
##      4:  51
##      5:  76
##     ---    
##  99996:  66
##  99997:  71
##  99998:  73
##  99999:  60
## 100000:  61
```

```
# These do not!
var_num <- 1
cohort[,var_num]
```

```
## Error in `[.data.table`(cohort, , var_num): j (the 2nd argument
inside [...]) is a single symbol but column name 'var_num' is not
found. Perhaps you intended DT[, ..var_num]. This difference to
data.frame is deliberate and explained in FAQ 1.1.
```

```
cohort[,..var_num] # works
```

```
##         patient_id
##      1:     100000
##      2:     100001
##      3:     100002
##      4:     100003
##      5:     100004
##     ---           
##  99996:     199995
##  99997:     199996
##  99998:     199997
##  99999:     199998
## 100000:     199999
```

```
var_name <- "age"
cohort[,var_name]
```

```
## Error in `[.data.table`(cohort, , var_name): j (the 2nd argument
inside [...]) is a single symbol but column name 'var_name' is not
found. Perhaps you intended DT[, ..var_name]. This difference to
data.frame is deliberate and explained in FAQ 1.1.

cohort[,..var_name] # works

##          age
##      1:   70
##      2:   64
##      3:   76
##      4:   51
##      5:   76
##     ---
##  99996:   66
##  99997:   71
##  99998:   73
##  99999:   60
## 100000:   61
```

More Complex j arguments

- As stated the j argument can be any valid R expression
- If we return a list of items, it will be coerced into a data.table
- We can name the items of the list, and it will be inherited to column names

```
## here is the average and SD of age
cohort[,mean(age)]

## [1] 64.93313

cohort[,sd(age)]

## [1] 10.00578

## Returned as a data.table
cohort[,list(mean(age),
             sd(age))]

##          V1       V2
## 1: 64.93313 10.00578

## Name each column in the result
cohort[,list(avg=mean(age),
             stdev=sd(age))]
```

```
##         avg    stdev
## 1: 64.93313 10.00578
```

The j expression

Question: Calculate the average home runs (hr variable) and return the result as a data.table Question: Calculate the total number of at bats (ab variable) and the total number of rbi's (rbi variable) and the total number of players with a team value of "HOU". Name these columns in the resulting data.table

- Answer

```
baseball_dt[,list(mean(hr))]
```

```
##          V1
## 1: 5.234204
```

```
baseball_dt[,list(tot_ab = sum(ab),
                  tot_rbi = sum(rbi,na.rm=T),
                  tot_hous = sum(team=="HOU"))]
```

```
##     tot_ab tot_rbi tot_hous
## 1: 4891061  641644      497
```

The j argument, complex expressions

- We can pass in more complex expressions in the j expression
- As before, if the result is not a simple list, it will not be a data.table
- This should remind you of thewith() command in base r dataframes

```
## here is a table of gender by mortality
cohort[,table(gender,mortality)]
```

```
##         mortality
## gender      No   Yes
##   Female  8399  1533
##   Male   73612 16456
```

```
## With row proportions
cohort[,prop.table(table(gender,mortality),1)]
```

```
##         mortality
## gender          No       Yes
##   Female 0.8456504 0.1543496
##   Male   0.8172936 0.1827064

## chi-square test on gender and mortality
cohort[,chisq.test(gender,mortality)]

##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  gender and mortality
## X-squared = 48.566, df = 1, p-value = 3.194e-12
```

The j expression, reducing results

- As an example of a more complex argument to j lets perform a logistic regression on this dataset

```
## logistic regression on mortality
mod_result_1 <-
  cohort[,glm(mortality == "Yes" ~ age + gender + SES,
              family = "binomial")]
summary(mod_result_1)

##
## Call:
## glm(formula = mortality == "Yes" ~ age + gender + SES, family =
"binomial")
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.9126  -0.6536  -0.6099  -0.5495   2.1987
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.9637216  0.0640235 -46.291  < 2e-16 ***
## age          0.0189530  0.0008296  22.846  < 2e-16 ***
## genderMale   0.2065857  0.0291515   7.087 1.37e-12 ***
## SESLow       0.0305045  0.0232105   1.314    0.189
## SESMed       0.0144423  0.0223347   0.647    0.518
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 94245  on 99999  degrees of freedom
## Residual deviance: 93667  on 99995  degrees of freedom
## AIC: 93677
##
## Number of Fisher Scoring iterations: 4
```

```
## all in one step
cohort[,summary(glm(mortality == "Yes" ~ age + gender + SES,
                    family = "binomial"))]
```

```
##
## Call:
## glm(formula = mortality == "Yes" ~ age + gender + SES, family =
"binomial")
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.9126  -0.6536  -0.6099  -0.5495   2.1987
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.9637216  0.0640235 -46.291  < 2e-16 ***
## age          0.0189530  0.0008296  22.846  < 2e-16 ***
## genderMale   0.2065857  0.0291515   7.087 1.37e-12 ***
## SESLow       0.0305045  0.0232105   1.314    0.189
## SESMed       0.0144423  0.0223347   0.647    0.518
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 94245  on 99999  degrees of freedom
## Residual deviance: 93667  on 99995  degrees of freedom
## AIC: 93677
##
## Number of Fisher Scoring iterations: 4
```

Using the j argument for variable transformations

- We may wish to have a transformed version of a variable

- We can name simply make a list of the variables we want, and add in the transformations

```
## Lets select patient_id, age, and add log age
cohort[,list(patient_id,
             age,
             l_age=log(age))]
```

```
##         patient_id age    l_age
##      1:     100000  70 4.248495
##      2:     100001  64 4.158883
##      3:     100002  76 4.330733
##      4:     100003  51 3.931826
##      5:     100004  76 4.330733
##     ---
##  99996:     199995  66 4.189655
##  99997:     199996  71 4.262680
##  99998:     199997  73 4.290459
##  99999:     199998  60 4.094345
## 100000:     199999  61 4.110874
```

Evaluating expressions by groups

- If we specify the `by=` argument, the `j` argument will be evaluated within each group specified

- This is an especially fast way to evaluate expressions by group levels

- The `by=` variable is automatically returned as part of the expression

```
## Average age by different gender levels
cohort[,mean(age),by=gender]
```

```
##    gender       V1
## 1:   Male 64.91831
## 2: Female 65.06756
```

```
## Use a named list to name the variable
cohort[,list(avg_age=mean(age)),
       by=gender]
```

```
##    gender  avg_age
## 1:   Male 64.91831
## 2: Female 65.06756
## Return multiple columns
cohort[,list(avg_age=mean(age),
             sd_age=sd(age)),
       by=gender]
```

```
##    gender  avg_age    sd_age
## 1:   Male 64.91831 10.007413
## 2: Female 65.06756  9.990478
```

Evaluating expressions by groups

> Question: For the rows after the year 2000, calculate the average and total rbi's for each team
> Question: For only the rows after the year 2000, calculate the average rbi's for each league variable (lg)

- Answer

```
baseball_dt[year > 2000,
            list(avg_rbi=mean(rbi,na.rm=T),
                 tot_rbi=sum(rbi,na.rm=T)),
            by=team]
```

```
##     team   avg_rbi tot_rbi
##  1:  CLE 31.435897    1226
##  2:  CHA 35.216216    1303
##  3:  BAL 30.270270    1120
##  4:  SEA 26.783333    1607
##  5:  BOS 18.507463    1240
##  6:  TEX 25.697674    1105
##  7:  TBA 27.555556     496
##  8:  NYA 20.189873    1595
##  9:  TOR 44.785714    1254
## 10:  DET 31.150000     623
## 11:  OAK 19.470588     331
## 12:  KCA 17.741935     550
## 13:  MIN  8.730769     227
## 14:  ANA  1.727273      19
## 15:  HOU 34.094340    1807
## 16:  NYN 18.658537    1530
## 17:  LAN 25.589286    1433
## 18:  COL 25.222222    1135
## 19:  ARI 26.507692    1723
## 20:  SLN 27.711111    1247
## 21:   SFN 35.312500    2260
## 22:   ATL 22.104167    1061
## 23:   MIL 27.384615     712
## 24:   PHI 14.318182     630
## 25:   MON 22.777778     205
## 26:   CHN 31.851852    1720
## 27:   PIT 16.818182     370
## 28:   FLO 21.193548     657
## 29:   SDN 19.044776    1276
## 30:   CIN 22.000000     770
## 31:   WAS 13.222222     119
## 32:   LAA 27.000000      54
##      team   avg_rbi tot_rbi
```

```
baseball_dt[year > 2000,
            list(avg_rbi=mean(rbi,na.rm=T),
                 tot_rbi=sum(rbi,na.rm=T)),
            by=lg]
```

```
##    lg  avg_rbi tot_rbi
## 1: AL 24.75728   12750
## 2: NL 24.70861   18655
```

Evaluating expressions by groups

- We may want to group by more than one level
- In that case, we can pass in either a list of column names as objects, or a character vector of column names
- If we pass in a list, we can create a new expression level in the list

```
## multiple by variables using list
cohort[,list(age=mean(age)),
       by=list(gender,SES)]
```

```
##    gender  SES      age
## 1:   Male  Med 64.83112
## 2:   Male  Low 64.95914
## 3:   Male High 65.04295
## 4: Female  Med 64.92555
## 5: Female High 65.23148
## 6: Female  Low 65.16029
## multiple by variables using character vector
cohort[,list(age=mean(age)),
       by=c("gender","SES")]
```

```
##    gender  SES      age
## 1:   Male  Med 64.83112
## 2:   Male  Low 64.95914
## 3:   Male High 65.04295
## 4: Female  Med 64.92555
## 5: Female High 65.23148
## 6: Female  Low 65.16029
```

```
## Use an expression for a by level
cohort[,list(age=mean(age)),
       by=list(gender,young=age<50)]
```

```
##    gender young      age
## 1:   Male FALSE 66.22100
## 2: Female FALSE 66.33298
## 3:   Male  TRUE 45.22405
## 4: Female  TRUE 44.99491
```

Sorting a Data.table

- We often wish to sort the results of an expression such as the one we just saw
- We can use `setorder()` command to sort a data.table in place
- Rather than set the order in place, we can also return the data.table in a new order by calling the order() command, then passing it into the i= argument.

```
res1 <-
  baseball_dt[year > 2000,
            list(avg_rbi=mean(rbi,na.rm=T),
                 tot_rbi=sum(rbi,na.rm=T)),
            by=team]
setorder(res1,avg_rbi)
res1
```

```
##     team   avg_rbi tot_rbi
##  1:  ANA  1.727273      19
##  2:  MIN  8.730769     227
##  3:  WAS 13.222222     119
##  4:  PHI 14.318182     630
##  5:  PIT 16.818182     370
##  6:  KCA 17.741935     550
##  7:  BOS 18.507463    1240
##  8:  NYN 18.658537    1530
##  9:  SDN 19.044776    1276
## 10:  OAK 19.470588     331
## 11:  NYA 20.189873    1595
## 12:  FLO 21.193548     657
## 13:  CIN 22.000000     770
## 14:  ATL 22.104167    1061
## 15:  MON 22.777778     205
## 16:  COL 25.222222    1135
## 17:  LAN 25.589286    1433
## 18:  TEX 25.697674    1105
## 19:  ARI 26.507692    1723
## 20:  SEA 26.783333    1607
## 21:  LAA 27.000000      54
## 22:  MIL 27.384615     712
## 23:  TBA 27.555556     496
## 24:  SLN 27.711111    1247
## 25:  BAL 30.270270    1120
## 26:  DET 31.150000     623
## 27:  CLE 31.435897    1226
## 28:  CHN 31.851852    1720
## 29:  HOU 34.094340    1807
## 30:  CHA 35.216216    1303
## 31:  SFN 35.312500    2260
## 32:  TOR 44.785714    1254
##     team   avg_rbi tot_rbi

## Same thing, but no intermediate object
## Does not order in place
res1 <-
  baseball_dt[year > 2000,
            list(avg_rbi=mean(rbi,na.rm=T),
                 tot_rbi=sum(rbi,na.rm=T)),
            by=team][order(avg_rbi)]
```

Regression by Group

- Any valid R functions can be evaluated by group....this is powerful!

- We can fit a multivariate regression in the `j` expression and extract the p-values associated with age and mortality

- The first step is to figure out how to get the result you want, then add the `by=` argument in to resolve the expression by group

- In this case, we are fitting a linear regression, then extracting the p-values associated with the mortality predictor

```
## extract the p-values for age and mortality
cohort[,summary(lm(age ~ gender + mortality))$coefficients[2,4],
       by=hosp_id]
```

```
##      hosp_id          V1
##   1:      77 0.121973347
##   2:      45 0.210289025
##   3:       5 0.980924814
##   4:      13 0.887124987
##   5:      70 0.858322923
##   6:      27 0.951418487
##   7:      29 0.491992103
##   8:      85 0.239406643
##   9:      41 0.797902663
##  10:      64 0.603953309
##  11:      43 0.042534060
##  12:      62 0.288284257
##  13:      54 0.795493702
##  14:      69 0.567321025
##  15:      20 0.645297833
##  16:      46 0.282262549
##  17:      19 0.457915985
##  18:       1 0.241606281
##  19:      81 0.567469864
##  20:      55 0.630498498
##  21:      88 0.054499752
##  22:      34 0.064525949
##  23:      15 0.086299501
##  24:      35 0.410806608
##  25:      94 0.557116047
##  26:       2 0.519355657
##  27:      76 0.589302422
##  28:      75 0.206815597
##  29:      66 0.583151953
##  30:      91 0.111223202
##  31:      63 0.502365243
##  32:       7 0.550516319
##  33:      22 0.285180780
##  34:      79 0.336277925
##  35:      90 0.951291640
##  36:      95 0.541878016
##  37:      97 0.963290970
##  38:       6 0.780778184
##  39:      36 0.949266143
##  40:      61 0.250958368
##  41:      82 0.566796959
##  42:      25 0.838208685
##  43:       4 0.231794037
##  44:      50 0.333371025
##  45:      33 0.245258738
##  46:      26 0.405456544
##  47:      52 0.961953487
##  48:      12 0.975255867
##  49:      14 0.835948923
##  50:      18 0.971313166
##  51:      31 0.569980012
##  52:      53 0.646286053
##  53:      78 0.886059765
##  54:      59 0.624216847
##  55:      28 0.252300795
##  56:      93 0.020239762
##  57:      87 0.373143181
##  58:      72 0.232146768
##  59:      60 0.298261161
##  60:      51 0.794773690
##  61:      17 0.722502499
##  62:       9 0.080573683
##  63:      73 0.722582454
##  64:      39 0.330502188
##  65:       8 0.518673141
##  66:     100 0.814276270
##  67:      67 0.222394088
##  68:      56 0.011362725
##  69:      48 0.257889098
##  70:      40 0.051644566
##  71:      98 0.331692019
##  72:      57 0.860790323
##  73:       80 0.681973253
##  74:        3 0.091395810
##  75:       58 0.012353405
##  76:       74 0.260627155
##  77:       44 0.299418585
##  78:       30 0.432579802
##  79:       92 0.680032805
##  80:       96 0.032909438
##  81:       47 0.573595675
##  82:       84 0.721069120
##  83:       89 0.080750311
##  84:       24 0.118326322
##  85:       49 0.277934332
##  86:       99 0.126635305
##  87:       21 0.812356905
##  88:       83 0.351053376
##  89:       23 0.688500971
##  90:       86 0.926979107
##  91:       42 0.816799942
##  92:       16 0.809076281
##  93:       38 0.158349413
##  94:       10 0.027490918
##  95:       11 0.467260926
##  96:       32 0.241127947
##  97:       37 0.953548590
##  98:       71 0.558666806
##  99:       68 0.009686422
## 100:       65 0.055406935
##      hosp_id          V1
```

Updating Values inside a Data.Table

- What if we want to add a column into a data.table or perhaps change an existing column?
- We use the `:=` operator in the `j` argument to update columns inside the data.table
- There are a few different ways to use this operator (next slides will show them)

```
## Create a new variable and save it to the DT
cohort[,l_age:=log(age)]
cohort
```

```
##         patient_id hosp_id age gender mortality  SES    l_age
##      1:     100000      77  70   Male       Yes  Med 4.248495
##      2:     100001      45  64   Male        No  Low 4.158883
##      3:     100002       5  76   Male        No  Med 4.330733
##      4:     100003      13  51   Male        No High 3.931826
##      5:     100004      70  76   Male        No  Low 4.330733
##     ---                                                      
##  99996:     199995      60  66   Male        No  Med 4.189655
##  99997:     199996      14  71   Male        No  Low 4.262680
##  99998:     199997      39  73   Male       Yes  Med 4.290459
##  99999:     199998      21  60 Female        No High 4.094345
## 100000:     199999      81  61   Male        No  Low 4.110874
```

```
## to erase the variable we assign it a NULL value
cohort[,l_age:=NULL]
cohort
```

```
##         patient_id hosp_id age gender mortality  SES
##      1:     100000      77  70   Male       Yes  Med
##      2:     100001      45  64   Male        No  Low
##      3:     100002       5  76   Male        No  Med
##      4:     100003      13  51   Male        No High
##      5:     100004      70  76   Male        No  Low
##     ---                                             
##  99996:     199995      60  66   Male        No  Med
##  99997:     199996      14  71   Male        No  Low
##  99998:     199997      39  73   Male       Yes  Med
##  99999:     199998      21  60 Female        No High
## 100000:     199999      81  61   Male        No  Low
```

```
names(cohort)
```

```
## [1] "patient_id" "hosp_id"    "age"         "gender"     "mortality"

## [6] "SES"
```

Update Multiple Columns

- If we want to update multiple columns, we change the syntax slightly
- Note that any existing column we reference is over-written, and any new column referenced will be created

```
## I use an if-else statement here to create the variable
cohort[,':='(l_age=log(age),
             age_cat=ifelse(age<55, "Young", "Less Young"))]
cohort

##         patient_id hosp_id age gender mortality  SES    l_age
age_cat
##      1:     100000      77  70   Male       Yes  Med 4.248495 Less
Young
##      2:     100001      45  64   Male        No  Low 4.158883 Less
Young
##      3:     100002       5  76   Male        No  Med 4.330733 Less
Young
##      4:     100003      13  51   Male        No High 3.931826
Young
##      5:     100004      70  76   Male        No  Low 4.330733 Less
Young
##     ---

##  99996:     199995      60  66   Male        No  Med 4.189655 Less
Young
##  99997:     199996      14  71   Male        No  Low 4.262680 Less
Young
##  99998:     199997      39  73   Male       Yes  Med 4.290459 Less
Young
##  99999:     199998      21  60 Female        No High 4.094345 Less
Young
## 100000:     199999      81  61   Male        No  Low 4.110874 Less
Young

## alternatively, we could have just written two separate lines
cohort[,l_age:=log(age)]
cohort[,age_cat:=ifelse(age<55, "Young", "Less Young")]
```

Combining i and j with :=

- If we use the i argument to subset the data then we are only updating the values in that subset when we use :=

```
## I can conditionally update values
## by combining the first two arguments
cohort[age > 70 & gender == "Male", risk_cat := "High"]
cohort[!(age > 70 & gender == "Male"), risk_cat := "Low"]
cohort
```

```
##         patient_id hosp_id age gender mortality  SES    l_age
age_cat
##      1:     100000      77  70   Male       Yes  Med 4.248495 Less
Young
##      2:     100001      45  64   Male        No  Low 4.158883 Less
Young
##      3:     100002       5  76   Male        No  Med 4.330733 Less
Young
##      4:     100003      13  51   Male        No High 3.931826
Young
##      5:     100004      70  76   Male        No  Low 4.330733 Less
Young
##     ---

##  99996:     199995      60  66   Male        No  Med 4.189655 Less
Young
##  99997:     199996      14  71   Male        No  Low 4.262680 Less
Young
##  99998:     199997      39  73   Male       Yes  Med 4.290459 Less
Young
##  99999:     199998      21  60 Female        No High 4.094345 Less
Young
## 100000:     199999      81  61   Male        No  Low 4.110874 Less
Young
##         risk_cat
##      1:      Low
##      2:      Low
##      3:     High
##      4:      Low
##      5:     High
##     ---
##  99996:      Low
##  99997:     High
##  99998:     High
##  99999:      Low
## 100000:      Low
```

Group by updates

- We often want to calculate a summary statistic within groups, and then join it back to the original data we calculated it from

- This is very easy by combining the `:=` operator in the `j` argument with the `by=` argument
- Here we calculate the average age within hospital, and assign it to the data.table

```
cohort[,avg_age_hosp:=mean(age,na.rm=T),
       by=hosp_id]
```

Exercise: Group by and updates

Question: Using the baseball data, answer this question: Starting after the year 2000, how many players (id denotes players) had an average rbi greater than 40?
Question: Calculate the average rbi value per team and year and save it back to the data.table

- Answer

```
## how many players had average rbi above 40
baseball_dt[year>2000,
            list(avg_rbi=mean(rbi,na.rm=T)),
            by=id][avg_rbi > 40]

##             id   avg_rbi
##   1:  bellda01  53.57143
##   2: bicheda01  49.00000
##   3: boonebr01  80.83333
##   4: burksel01  48.50000
##   5: cansejo01  49.00000
##   6: castivi02  56.50000
##   7: coninje01  47.40000
##   8: delgaca01 110.00000
##   9: evereca01  45.87500
##  10: gonzaju03  52.40000
##  11: justida01  50.00000
##  12: martied01  84.00000
##  13: martiti02  76.40000
##  14: mcgrifr01  50.40000
##  15: olerujo01  60.83333
##  16: oneilpa01  70.00000
##  17: palmera01  97.60000
##  18: ramirma02 114.28571
##  19: ripkeca01  68.00000
##  20: rodriiv01  68.28571
## 21: thomafr04  70.14286
## 22: thomeji01 101.85714
## 23: valenjo03  54.42857
## 24: vizquom01  50.57143
## 25: willibe02  75.83333
## 26:  aloumo01  78.85714
## 27: bagweje01  87.20000
## 28: biggicr01  62.00000
## 29: bondsba01  84.42857
## 30: burnije01  68.14286
## 31: edmonji01  86.42857
## 32: finlest01  52.75000
## 33: floydcl01  55.55556
## 34: gonzalu01  88.14286
## 35: gracema01  47.33333
## 36: greensh01  74.37500
## 37: griffke02  61.57143
## 38: grissma02  60.80000
## 39: jordabr01  43.66667
## 40:  kentje01  95.14286
## 41: kleskry01  63.57143
## 42: lopezja01  56.71429
## 43: mcgwima01  64.00000
## 44: nevinph01  50.77778
## 45: piazzmi01  64.85714
## 46: sandere02  63.28571
## 47: sheffga01  94.28571
## 48:  snowjt01  40.33333
## 49:  sosasa01  98.00000
## 50: stairma01  45.11111
## 51: venturo01  47.40000
## 52: walkela01  67.50000
## 53: walketo04  41.22222
## 54: whitede03  47.00000
## 55: whitero02  47.12500
## 56: willima04  40.33333
## 57: zeileto01  45.20000
##            id   avg_rbi
```

```
## calculate the average rbi by team and year and save it
baseball_dt[,avg_rbi_team_yr:=mean(rbi,na.rm=T),
            by=list(team,year)]
```

.N variable and Copying

- Data.table is careful to not make extra copies (part of the efficiency)

- If you want to make another copy of a data.table, you should use the `copy()` command explicitly
- Also, there is a special variable called `.N` that can be called in the `j` expression. This is the length of the current group, which can be used to quickly calculate the total rows in a group

```
## Make a copy of a data.table
cohort2 <- copy(cohort)

## Use .N
cohort[,list(avg=mean(age,na.rm=T),.N),
       by=gender]
```

```
##    gender      avg     N
## 1:   Male 64.91831 90068
## 2: Female 65.06756  9932
```

Setting Keys and Data.table

- We said one reason to use data.table was computational speed

- Data.table has speed gains primarily due to leveraging sorted data to perform binary scans

- This concept is exactly like a telephone book. You can quickly find someone in the telephone book because it is sorted by last name then by first name. That is an example of a binary search. A slower way to do the same search would be to check every name in the phone book to see if it matches the one you are looking for (vector scan)

- In order to leverage a binary search, we must first sort the data.table (just like the phone book must be sorted by last name then first name)

Setting Keys and Data.table

- We call this sorting "setting the keys" and the keys of the data.table are the variables we sorted it by

- We can sort the table, or "set the keys" with the `setkey()` command

- If we want descending sort order, we need to use the `setorder()` command

```
## Set the key to be hosp_id
setkey(cohort,hosp_id)
## notice it is sorted now
cohort

##         patient_id hosp_id age gender mortality  SES    l_age
age_cat
##      1:     100017       1  70   Male        No  Med 4.248495 Less
Young
##      2:     100049       1  64   Male       Yes High 4.158883 Less
Young
##      3:     100094       1  71   Male        No  Med 4.262680 Less
Young
##      4:     100210       1  72   Male        No  Low 4.276666 Less
Young
##      5:     100217       1  75   Male        No  Med 4.317488 Less
Young
##     ---

##  99996:     199342     100  71 Female        No  Med 4.262680 Less
Young
##  99997:     199778     100  57   Male        No  Med 4.043051 Less
Young
##  99998:     199925     100  72   Male        No  Low 4.276666 Less
Young
##  99999:     199952     100  82   Male        No High 4.406719 Less
Young
## 100000:     199976     100  60 Female        No  Low 4.094345 Less
Young
##         risk_cat avg_age_hosp
##      1:      Low     65.08249
##      2:      Low     65.08249
##      3:     High     65.08249
##      4:     High     65.08249
##      5:     High     65.08249
##     ---
##  99996:      Low     64.25282
##  99997:      Low     64.25282
##  99998:     High     64.25282
##  99999:     High     64.25282
## 100000:      Low     64.25282
```

```
## Set the key to be hosp_id, then gender
setkey(cohort,hosp_id,gender)
## Set the order to be gender, then descending age
setorder(cohort,gender,-age)[]
```

```
##          patient_id hosp_id age gender mortality  SES    l_age
age_cat
##      1:     118862      11  98 Female        No  Low 4.584967 Less
Young
##      2:     173163      17  96 Female        No  Low 4.564348 Less
Young
##      3:     174555      23  96 Female        No  Low 4.564348 Less
Young
##      4:     139345      28  96 Female        No  Low 4.564348 Less
Young
##      5:     128902      56  96 Female        No  Low 4.564348 Less
Young
##     ---

##  99996:     181651      66  27   Male        No  Low 3.295837
Young
##  99997:     130714      31  26   Male        No  Med 3.258097
Young
##  99998:     166410       4  25   Male       Yes  Med 3.218876
Young
##  99999:     108941      44  23   Male       Yes High 3.135494
Young
## 100000:     102078      38  22   Male        No  Low 3.091042
Young
##          risk_cat avg_age_hosp
##      1:      Low     64.75764
##      2:      Low     65.10287
##      3:      Low     64.64581
##      4:      Low     65.27697
##      5:      Low     64.95286
##     ---
##  99996:      Low     64.87476
##  99997:      Low     64.52484
##  99998:      Low     64.95838
##  99999:      Low     64.96362
## 100000:      Low     65.51812
```

Leveraging the keys for speed gains

- Once a key has been set in a data.table, we can use that key to make many groupwise operations on the data.table faster
- The first example is subsetting the data.table. Recall we use the `i` argument to subset the data.table. If we pass in a list or a data.table to the `i` argument, it will join to the keys of the data.table effectively subsetting the data.
- an alternative is to use the shortcut function `J()`

```
setkey(cohort,gender)
## subset down to hosp_id == 11
## fast way leveraging the keys, all equivalent
cohort[list("Male")]
```

```
##        patient_id hosp_id age gender mortality  SES    l_age
age_cat
##     1:     126803      41 109   Male        No  Low 4.691348 Less
Young
##     2:     119396      35 108   Male        No  Med 4.682131 Less
Young
##     3:     114797      61 107   Male       Yes  Med 4.672829 Less
Young
##     4:     110129      72 107   Male        No  Med 4.672829 Less
Young
##     5:     156709      54 105   Male        No High 4.653960 Less
Young
##    ---

## 90064:     181651      66  27   Male        No  Low 3.295837
Young
## 90065:     130714      31  26   Male        No  Med 3.258097
Young
## 90066:     166410       4  25   Male       Yes  Med 3.218876
Young
## 90067:     108941      44  23   Male       Yes High 3.135494
Young
```

```
## 90068:     102078      38  22   Male         No  Low 3.091042
Young
##        risk_cat avg_age_hosp
##     1:     High     64.72016
##     2:     High     65.03490
##     3:     High     65.21696
##     4:     High     65.30579
##     5:     High     64.78392
##    ---
## 90064:      Low     64.87476
## 90065:      Low     64.52484
## 90066:      Low     64.95838
## 90067:      Low     64.96362
## 90068:      Low     65.51812
```

```
cohort[.("Male")]
```

```
##        patient_id hosp_id age gender mortality  SES    l_age
age_cat
##     1:     126803      41 109   Male         No  Low 4.691348 Less
Young
##     2:     119396      35 108   Male         No  Med 4.682131 Less
Young
##     3:     114797      61 107   Male        Yes  Med 4.672829 Less
Young
##     4:     110129      72 107   Male         No  Med 4.672829 Less
Young
##     5:     156709      54 105   Male         No High 4.653960 Less
Young
##    ---

## 90064:     181651      66  27   Male         No  Low 3.295837
Young
## 90065:     130714      31  26   Male         No  Med 3.258097
Young
## 90066:     166410       4  25   Male        Yes  Med 3.218876
Young
## 90067:     108941      44  23   Male        Yes High 3.135494
Young
## 90068:     102078      38  22   Male         No  Low 3.091042
Young
##        risk_cat avg_age_hosp
##     1:     High     64.72016
##      2:     High     65.03490
##      3:     High     65.21696
##      4:     High     65.30579
##      5:     High     64.78392
##    ---
## 90064:      Low     64.87476
## 90065:      Low     64.52484
## 90066:      Low     64.95838
## 90067:      Low     64.96362
## 90068:      Low     65.51812
```

```
cohort[J("Male")]
```

```
##        patient_id hosp_id age gender mortality  SES    l_age
age_cat
##      1:     126803      41 109   Male        No  Low 4.691348 Less
Young
##      2:     119396      35 108   Male        No  Med 4.682131 Less
Young
##      3:     114797      61 107   Male       Yes  Med 4.672829 Less
Young
##      4:     110129      72 107   Male        No  Med 4.672829 Less
Young
##      5:     156709      54 105   Male        No High 4.653960 Less
Young
##    ---

## 90064:     181651      66  27   Male        No  Low 3.295837
Young
## 90065:     130714      31  26   Male        No  Med 3.258097
Young
## 90066:     166410       4  25   Male       Yes  Med 3.218876
Young
## 90067:     108941      44  23   Male       Yes High 3.135494
Young
## 90068:     102078      38  22   Male        No  Low 3.091042
Young
##        risk_cat avg_age_hosp
##      1:     High     64.72016
##      2:     High     65.03490
##      3:     High     65.21696
##      4:     High     65.30579
##      5:     High     64.78392
##    ---
## 90064:       Low     64.87476
## 90065:       Low     64.52484
## 90066:       Low     64.95838
## 90067:       Low     64.96362
## 90068:       Low     65.51812
```

```
## slow way (vector scan)
cohort[gender == "Male"]
```

```
##        patient_id hosp_id age gender mortality  SES    l_age
age_cat
##     1:     126803      41 109   Male        No  Low 4.691348 Less
Young
##     2:     119396      35 108   Male        No  Med 4.682131 Less
Young
##     3:     114797      61 107   Male       Yes  Med 4.672829 Less
Young
##     4:     110129      72 107   Male        No  Med 4.672829 Less
Young
##     5:     156709      54 105   Male        No High 4.653960 Less
Young
##    ---

## 90064:     181651      66  27   Male        No  Low 3.295837
Young
## 90065:     130714      31  26   Male        No  Med 3.258097
Young
## 90066:     166410       4  25   Male       Yes  Med 3.218876
Young
## 90067:     108941      44  23   Male       Yes High 3.135494
Young
## 90068:     102078      38  22   Male        No  Low 3.091042
Young
##        risk_cat avg_age_hosp
##     1:     High     64.72016
##     2:     High     65.03490
##     3:     High     65.21696
##     4:     High     65.30579
##     5:     High     64.78392
##    ---
## 90064:      Low     64.87476
## 90065:      Low     64.52484
## 90066:       Low     64.95838
## 90067:       Low     64.96362
## 90068:       Low     65.51812
```

Subsetting with multiple keys

- If we have set multiple keys to the data.table, we can pass in multiple arguments to i for the fast subset

```
## subset down to the males with mortality == Yes
setkey(cohort,gender,mortality)
cohort[J("Male","Yes")]
```

```
##          patient_id hosp_id age gender mortality  SES    l_age
age_cat
##      1:      114797      61 107   Male       Yes  Med 4.672829 Less
Young
##      2:      191912      73 103   Male       Yes  Med 4.634729 Less
Young
##      3:      108425      64 102   Male       Yes High 4.624973 Less
Young
##      4:      157047      46 101   Male       Yes High 4.615121 Less
Young
##      5:      104316      48 100   Male       Yes  Low 4.605170 Less
Young
##    ---

## 16452:      117968       1  31   Male       Yes  Med 3.433987
Young
## 16453:      176867      71  31   Male       Yes  Med 3.433987
Young
## 16454:      127488      95  29   Male       Yes  Low 3.367296
Young
## 16455:      166410       4  25   Male       Yes  Med 3.218876
Young
## 16456:      108941      44  23   Male       Yes High 3.135494
Young
##          risk_cat avg_age_hosp
##      1:      High     65.21696
##      2:      High     65.13546
##      3:      High     64.81524
##      4:      High     64.97073
##      5:      High     65.06635
##    ---
## 16452:      Low     65.08249
## 16453:      Low     64.31027
## 16454:      Low     64.82656
## 16455:      Low     64.95838
## 16456:      Low     64.96362
```

Exercise

> Question: Using the Baseball data, perform a fast subset (set the key, then use J) of the data only looking at the league NL (lg variable)

- Answer

```
setkey(baseball_dt,lg)
baseball_dt[J("NL")]
```

```
##                 id year stint team lg   g  ab  r   h X2b X3b hr rbi
sb cs bb so
##     1: ansonca01 1876     1  CHN NL  66 309 63 110   9   7  2  59
NA NA 12  8
##     2: burdoja01 1876     1  HAR NL  69 309 66  80   9   1  0  23
NA NA 13 16
##     3: forceda01 1876     1  PHN NL  60 284 48  66   6   0  0  17
NA NA  5  3
##     4: forceda01 1876     2  NY3 NL   1   3  0   0   0   0  0   0
NA NA  0  0
##     5: gerhajo01 1876     1  LS1 NL  65 292 33  76  10   3  2  18
NA NA  3  5
##    ---

## 11374: benitar01 2007     2  FLO NL  34   0  0   0   0   0  0   0
0  0  0  0
## 11375: benitar01 2007     1  SFN NL  19   0  0   0   0   0  0   0
0  0  0  0
## 11376: ausmubr01 2007     1  HOU NL 117 349 38  82  16   3  3  25
6  1 37 74
## 11377:  aloumo01 2007     1  NYN NL  87 328 51 112  19   1 13  49
3  0 27 30
## 11378: alomasa02 2007     1  NYN NL   8  22  1   3   1   0  0   0
0  0  0  3
##        ibb hbp sh sf gidp avg_rbi_team_yr
##     1:  NA  NA NA NA   NA        59.33333
##     2:  NA  NA NA NA   NA        31.00000
##     3:  NA   NA NA NA    NA         24.00000
##     4:  NA   NA NA NA    NA         10.00000
##     5:  NA   NA NA NA    NA         13.50000
##    ---
## 11374:   0    0  0  0     0          0.00000
## 11375:   0    0  0  0     0         34.20000
## 11376:   3    6  4  1    11         19.25000
## 11377:   5    2  0  3    13         20.25000
## 11378:   0    0  0  0     0         20.25000
```

Group by calculations on the key

- Doing group by calculations on a data.table when the group is the key is faster.
- Thus for very large operations, you may set the key first, then do the group by calculations on the data.table.
- This requires doing a join by passing in a data.table or list to the i argument
- You need to say `by=.EACHI` to get this behaviour.

```
## mortality
cohort[,sum(mortality == "Yes")/.N]
```

```
## [1] 0.17989
```

```
## mortality by hospital ID (slower way)
cohort[,sum(mortality == "Yes")/.N,
       by=hosp_id][1:10]
```

```
##     hosp_id        V1
##  1:      11 0.1832512
##  2:      17 0.1668211
##  3:      23 0.1897074
##  4:      28 0.1584062
##  5:      56 0.2036108
##  6:      73 0.1962151
##  7:      29 0.2007913
##  8:      45 0.1761711
##  9:      60 0.1730580
## 10:      64 0.1961905
```

```
## For an even faster operation, use the key
## set key
setkey(cohort,hosp_id)
## mortality by hosp id, faster version of code
```

```
cohort[J(unique(hosp_id)),
       .(mortality_rate=sum(mortality == "Yes")/.N),
       by=.EACHI][1:10]
```

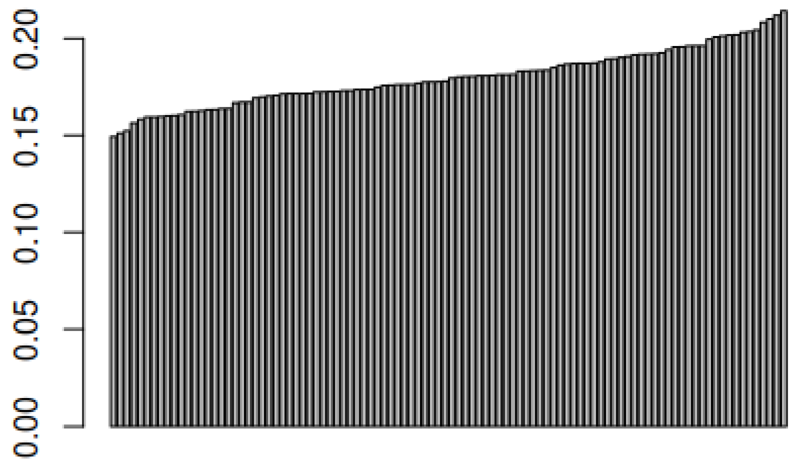
```
##     hosp_id mortality_rate
##  1:       1      0.1810865
##  2:       2      0.1815320
##  3:       3      0.1776181
##  4:       4      0.2121827
##  5:       5      0.1919087
##  6:       6      0.2141434
##  7:       7      0.1633919
##  8:       8      0.2102161
##  9:       9      0.1836529
## 10:      10      0.1715177
```

Exercise: Mortality Rates

Question: Using the cohort data, calculate the mortality rates by hospital id. Set the key of the table to hospital id before doing the group by calculation. The mortality rate should be the sum of the mortality equal to Yes values divided by the number of rows for that hospital (use the `.N` variable for size)
Question: After doing these calculations, create a barplot of this result. Before creating the barplot, sort the result by mortality rate from low to high

- Answer

```
setkey(cohort,hosp_id)
res1 <-
  cohort[,list(mort_rate=sum(mortality == "Yes")/.N),
         by=hosp_id]
setorder(res1,mort_rate)
res1[,barplot(mort_rate)]
```



```
##         [,1]
##   [1,]   0.7
##   [2,]   1.9
##   [3,]   3.1
##   [4,]   4.3
##   [5,]   5.5
##   [6,]   6.7
##   [7,]   7.9
##   [8,]   9.1
##   [9,]  10.3
##  [10,]  11.5
##  [11,]  12.7
##  [12,]  13.9
##  [13,]  15.1
##  [14,]  16.3
##  [15,]  17.5
##  [16,]  18.7
##  [17,]  19.9
##  [18,]  21.1
##  [19,]  22.3
##  [20,]  23.5
##  [21,]  24.7
##  [22,]  25.9
## [23,] 27.1
## [24,] 28.3
## [25,] 29.5
## [26,] 30.7
## [27,] 31.9
## [28,] 33.1
## [29,] 34.3
## [30,] 35.5
## [31,] 36.7
## [32,] 37.9
## [33,] 39.1
## [34,] 40.3
## [35,] 41.5
## [36,] 42.7
## [37,] 43.9
## [38,] 45.1
## [39,] 46.3
## [40,] 47.5
## [41,] 48.7
## [42,] 49.9
## [43,] 51.1
## [44,] 52.3
## [45,] 53.5
## [46,] 54.7
## [47,] 55.9
## [48,] 57.1
## [49,] 58.3
## [50,] 59.5
## [51,] 60.7
## [52,] 61.9
## [53,] 63.1
## [54,] 64.3
## [55,] 65.5
## [56,] 66.7
## [57,] 67.9
## [58,] 69.1
## [59,] 70.3
## [60,] 71.5
## [61,] 72.7
## [62,] 73.9
## [63,] 75.1
## [64,] 76.3
## [65,] 77.5
##  [66,]  78.7
##  [67,]  79.9
##  [68,]  81.1
##  [69,]  82.3
##  [70,]  83.5
##  [71,]  84.7
##  [72,]  85.9
##  [73,]  87.1
##  [74,]  88.3
##  [75,]  89.5
##  [76,]  90.7
##  [77,]  91.9
##  [78,]  93.1
##  [79,]  94.3
##  [80,]  95.5
##  [81,]  96.7
##  [82,]  97.9
##  [83,]  99.1
##  [84,] 100.3
##  [85,] 101.5
##  [86,] 102.7
##  [87,] 103.9
##  [88,] 105.1
##  [89,] 106.3
##  [90,] 107.5
##  [91,] 108.7
##  [92,] 109.9
##  [93,] 111.1
##  [94,] 112.3
##  [95,] 113.5
##  [96,] 114.7
##  [97,] 115.9
##  [98,] 117.1
##  [99,] 118.3
## [100,] 119.5
```

Data.table and the i argument

- When we pass a list or data.table or `J` into the `i` argument, we are technically performing a join which results in subset (like in the previous slides)

- We can specify which of the joined rows are returned with the `mult=` argument

- The options for `mult=` include “all” (the default), “first” and “last”

```
## sort by hospital id then age
setkey(cohort,hosp_id,age)
## join all the rows in hospitals one to three
cohort[J(c(1,2,3))]

##       patient_id hosp_id age gender mortality  SES    l_age
age_cat risk_cat
##    1:     159292       1  31   Male        No  Med 3.433987
Young      Low
##    2:     117968       1  31   Male       Yes  Med 3.433987
Young      Low
##    3:     139877       1  34   Male        No  Med 3.526361
Young      Low
##    4:     112040       1  36 Female        No  Med 3.583519
Young      Low
##    5:     135266       1  36   Male        No  Med 3.583519
Young      Low
##   ---

## 2917:     113069       3  92   Male        No  Med 4.521789 Less
Young     High
## 2918:     190428       3  92   Male        No  Med 4.521789 Less
Young     High
## 2919:     172256       3  95   Male        No High 4.553877 Less
Young     High
## 2920:     185063       3  95   Male        No  Med 4.553877 Less
Young     High
## 2921:     192823       3  95   Male        No High 4.553877 Less
Young     High
##       avg_age_hosp
##    1:     65.08249
##    2:     65.08249
##    3:     65.08249
##    4:     65.08249
##    5:     65.08249
##   ---
## 2917:     65.24230
## 2918:     65.24230
## 2919:     65.24230
## 2920:     65.24230
## 2921:     65.24230

## join only to the first row
## youngest person per hospital
cohort[J(c(1,2,3)),mult="first"]

##    patient_id hosp_id age gender mortality  SES    l_age age_cat
risk_cat
## 1:     159292       1  31   Male        No  Med 3.433987   Young
Low
## 2:     158174       2  33   Male        No  Low 3.496508   Young
Low
## 3:     103987       3  34   Male        No High 3.526361   Young
Low
##    avg_age_hosp
## 1:     65.08249
## 2:     65.09864
## 3:     65.24230

## join only to the last row
## oldest person per hospital
cohort[J(c(1,2,3)),mult="last"]

##    patient_id hosp_id age gender mortality  SES    l_age    age_cat
risk_cat
## 1:     175169       1  98   Male        No  Med 4.584967 Less Young
High
## 2:     100971       2 100   Male        No  Low 4.605170 Less Young
High
## 3:     192823       3  95   Male        No High 4.553877 Less Young
High
##    avg_age_hosp
## 1:     65.08249
## 2:     65.09864
## 3:     65.24230
```

Joins in Data.table

- We have already seen an example of a join in data.table when leveraging the keys and using `J` in the `i` argument

- We can join two data.tables together using the `i` argument, and the keys dictate which columns are joined

```
## read in hospital traits for example joins
hosp <- fread("hosp_demo.csv")
## set keys for both tables (these are the things we will merge on)
setkey(hosp,hosp_id)
setkey(cohort,hosp_id)

## right join: all the hospitals and whatever patients match
cohort[hosp]
```

```
##         patient_id hosp_id age gender mortality SES    l_age
age_cat
##      1:     159292       1  31   Male        No Med 3.433987
Young
##      2:     117968       1  31   Male       Yes Med 3.433987
Young
##      3:     139877       1  34   Male        No Med 3.526361
Young
##      4:     112040       1  36 Female        No Med 3.583519
Young
##      5:     135266       1  36   Male        No Med 3.583519
Young
##     ---

##  99996:     132099     100  89   Male        No Med 4.488636 Less
Young
##  99997:     105295     100  90   Male        No Low 4.499810 Less
Young
##  99998:     184176     100  91   Male        No Low 4.510860 Less
Young
##  99999:     192159     100  94   Male        No Med 4.543295 Less
Young
## 100000:     143580     100  98   Male       Yes Low 4.584967 Less
Young
##         risk_cat avg_age_hosp academic
##      1:      Low     65.08249      Yes
##      2:      Low     65.08249      Yes
##      3:      Low     65.08249      Yes
##      4:      Low     65.08249      Yes
##      5:      Low     65.08249      Yes
##     ---
##  99996:     High     64.25282       No
##  99997:     High     64.25282       No
##  99998:     High     64.25282       No
##  99999:     High     64.25282       No
## 100000:     High     64.25282       No
```

```
## left join: all the patients and whatever hospitals match
hosp[cohort]
```

```
##          hosp_id academic patient_id age gender mortality SES
l_age
##      1:        1      Yes     159292  31   Male        No Med
3.433987
##      2:        1      Yes     117968  31   Male       Yes Med
3.433987
##      3:        1      Yes     139877  34   Male        No Med
3.526361
##      4:        1      Yes     112040  36 Female        No Med
3.583519
##      5:        1      Yes     135266  36   Male        No Med
3.583519
##     ---

##  99996:      100       No     132099  89   Male        No Med
4.488636
##  99997:      100       No     105295  90   Male        No Low
4.499810
##  99998:      100       No     184176  91   Male        No Low
4.510860
##  99999:      100       No     192159  94   Male        No Med
4.543295
## 100000:      100       No     143580  98   Male       Yes Low
4.584967
##             age_cat risk_cat avg_age_hosp
##      1:       Young      Low     65.08249
##      2:       Young      Low     65.08249
##      3:       Young      Low     65.08249
##      4:       Young      Low     65.08249
##      5:       Young      Low     65.08249
##     ---
##  99996: Less Young     High     64.25282
##  99997: Less Young     High     64.25282
##  99998: Less Young     High     64.25282
##  99999: Less Young     High     64.25282
## 100000: Less Young     High     64.25282
```

```r
## inner join: only the overlapping hospitals and patients
hosp[cohort,nomatch=0]
```

```
##         hosp_id academic patient_id age gender mortality SES
l_age
##      1:       1      Yes     159292  31   Male        No Med
3.433987
##      2:       1      Yes     117968  31   Male       Yes Med
3.433987
##      3:       1      Yes     139877  34   Male        No Med
3.526361
##      4:       1      Yes     112040  36 Female        No Med
3.583519
##      5:       1      Yes     135266  36   Male        No Med
3.583519
##     ---

##  99996:     100       No     132099  89   Male        No Med
4.488636
##  99997:     100       No     105295  90   Male        No Low
4.499810
##  99998:     100       No     184176  91   Male        No Low
4.510860
##  99999:     100       No     192159  94   Male        No Med
4.543295
## 100000:     100       No     143580  98   Male       Yes Low
4.584967
##            age_cat risk_cat avg_age_hosp
##      1:      Young      Low     65.08249
##      2:      Young      Low     65.08249
##      3:      Young      Low     65.08249
##      4:      Young      Low     65.08249
##      5:      Young      Low     65.08249
##     ---
##  99996: Less Young     High     64.25282
##  99997: Less Young     High     64.25282
##  99998: Less Young     High     64.25282
##  99999: Less Young     High     64.25282
## 100000: Less Young     High     64.25282
## Full join (all patients and hospitals, even if they don't match)
merge(hosp,cohort,all=TRUE)
```

```
##         hosp_id academic patient_id age gender mortality SES
l_age
##      1:       1      Yes     159292  31   Male        No Med
3.433987
##      2:       1      Yes     117968  31   Male       Yes Med
3.433987
##      3:       1      Yes     139877  34   Male        No Med
3.526361
##      4:       1      Yes     112040  36 Female        No Med
3.583519
##      5:       1      Yes     135266  36   Male        No Med
3.583519
##     ---

##  99996:     100       No     132099  89   Male        No Med
4.488636
##  99997:     100       No     105295  90   Male        No Low
4.499810
##  99998:     100       No     184176  91   Male        No Low
4.510860
##  99999:     100       No     192159  94   Male        No Med
4.543295
## 100000:     100       No     143580  98   Male       Yes Low
4.584967
##            age_cat risk_cat avg_age_hosp
##      1:      Young      Low     65.08249
##      2:      Young      Low     65.08249
##      3:      Young      Low     65.08249
##      4:      Young      Low     65.08249
##      5:      Young      Low     65.08249
##     ---
##  99996: Less Young     High     64.25282
##  99997: Less Young     High     64.25282
##  99998: Less Young     High     64.25282
##  99999: Less Young     High     64.25282
## 100000: Less Young     High     64.25282
```

Conclusion

- Data.table functionality we covered

- Quickly read in files with `fread`
- Use the square brackets to interact with the data.table
- The first `i` argument subsets the data or joins if a data.table is given
- The second `j` argument accepts any valid r expressions
- If `j` resolves to a simple list a data.table is returned
- We can update or add columns using the `:=` operator
- We can do any of these operations by group using the `by=` command
- Data.table is faster than base R in many tasks
- Data.table uses memory more efficiently by making less copies of datasets as you work with them
- Data.table is one of the fastest ways to merge and work with big data in R