Datacamp data.table Class Exercises

Trenton Potgieter

Sunday, November 02, 2014

Chapter 1

Exercise 1 - Create and Subset a data.table

Instructions

- Create a data.table **my_first_data_table** with a column x = c("a","b","c","d","e") and a column y = c(1,2,3,4,5). Use data.table().
- Create a two-column data.table called DT that contains the four integers 1,2,1,2 in the first column a and the letters A,B,C,D in the second column b. Use recycling.
- Select the third row of DT and just print the result to the console.
- Select the second and third rows without using any comma at all and print the result to the console.

```r
library("data.table")

# Create your first data.table
my_first_data_table <- data.table(x = c("a","b","c","d","e"),y = c(1,2,3,4,5))

# Create a data.table using recycling
DT <- data.table(a = c(1L,2L), ## Note the "L" for integers
                 b = LETTERS[1:4]) ## Note "LETTERs" so no need to do this:
## b = c("A", "B", "C", "D")

# Print the third row to the console
DT[3]
```

```
##    a b
## 1: 1 C
```

```r
# Print the second and third row to the console, but do not use any comma at all
DT[2:3]
```

```
##    a b
## 1: 2 B
## 2: 1 C
```

Exercise 2 - Get to know data.table

Remember that the special symbol **.N** contains the number of rows. You can put this symbol inside square brackets

Instructions

- Select the penultimate row of the table. Make use of .N for this.
- Return the column names of the data.table.
- Return the number of rows and number of columns of the data.table.
- Select row 2 twice and row 3, returning a data.table with three rows where row 2 is a duplicate of row 1.

```
# Print the penultimate row of DT using `.N`
DT[.N-1]
```

```
##    a b
## 1: 1 C
```

```
# Print the column names of DT, and number of rows and number of columns
names(DT) ## or colnames(DT)
```

```
## [1] "a" "b"
```

```
dim(DT)
```

```
## [1] 4 2
```

```
# Select row 2 twice and row 3, returning a data.table with three rows where row 2 is a duplicate of ro
DT[c(2,2,3)]
```

```
##    a b
## 1: 2 B
## 2: 2 B
## 3: 1 C
```

Exercise 3 - Subsetting data tables

Remember the phrase "Take DT, subset rows using **i**, then calculate **j** grouped by **by**". Also, **j** can be used to select columns by wrapping the column names in **.()**. In addition to selecting columns, you can also call functions on them as if the columns were variables.

Instructions

- Create a subset columns B and C for rows 1 and 3, and print the result to the console.
- Assign to ans a data.table that contains two columns: B and val, where val is the product of A and C.
- Assign to answ another data.table answ <- DT[, .(B, val = **BLANK**)]. Fill in the blanks such that answ equals data.table(B=c("a", "b", "c", "d", "e", "a", "b", "c", "d", "e"), val = as.integer(c(6,7,8,9,10,1,2,3,4,5))).

```
# Create the data,table DT
DT <- data.table(A = 1:5, B = c("a", "b", "c", "d", "e"), C = 6:10)

# Subset rows 1 and 3, and columns B and C
DT[c(1, 3), .(B, C)]
```

```
##    B C
## 1: a 6
## 2: c 8
```

```
# Assign to ans the correct value
ans <- data.table(B = DT[, B], val = DT[,A * C])
# Note: Official solution is ans <- DT[, .(B, val = A * C)]

# Fill in the blank
answ <- DT[, .(B, val = c(C, A))]
```

Exercise 4 - The by basics

Remember if you supply a **j** expression and a by list of expressions, the **j** expression is repeated for each **by** group.

Instructions

- Convert the iris data set to a data.table called DT.
- For each Species, what is the mean Sepal.Length?
- Do exactly the same as in the instruction above, but this time group by the first letter of the Species name instead.

```
# load the iris dataset
data(iris)

# iris as data.table
DT <- as.data.table(iris)

# mean `Sepal.Length`
DT[, mean(Sepal.Length), by = .(Species)]
```

```
##       Species    V1
## 1:     setosa 5.006
## 2: versicolor 5.936
## 3:  virginica 6.588
```

```
# Group by the first letter
DT[, mean(Sepal.Length), by = substring(Species, 1, 1)]
```

```
##    substring    V1
## 1:         s 5.006
## 2:         v 6.262
```

```
# Note: substring(x, start, stop) extracts or replaces the substrings in a
# character vector.
```

Exercise 5 - Using .N and by

Remember that **.N** can be used in **i** and that it designates the number of rows in a data.table. There, it is typically used for returning the last row or an offset from it. **.N** can be used in **j** too and designates the number of rows in this group. The latter is especially powerful when you can use it in combination with by.

Instructions

- Group the specimens by Sepal area (to the nearest 10 cm2) and count how many occur in each group.
- Using the answer to the above question, you can now name the group Area and the count Count.

```
# Use the data.table DT for the iris dataset that was used in Exercise 4.

# Group the specimens by Sepal area (to the nearest 10 cm2) and count how many
# occur in each group. For example: DT[,___, by=10*___(___*___ / 10)]
DT[, .N, by = 10 * round(Sepal.Length * Sepal.Width/10)]
```

```
##    round   N
## 1:    20 117
## 2:    10  29
## 3:    30   4
```

```
# Now name the output columns `Area` and `Count`
DT[, .(Count = .N), by = .(Area = 10 * round(Sepal.Length * Sepal.Width/10))]
```

```
##    Area Count
## 1:   20   117
## 2:   10    29
## 3:   30     4
```

Exercise 6 - Return multiple numbers in j.

In the previous exercises we returned single numbers in **j**. However, this is not necessary, because you do not have to return only single numbers in **j**.

Instructions

- Given a new DT, calculate cumulative sum of C in column C while you group by A,B. Store it in a new data.table DT2 with 3 columns A,B and C.
- Select from DT2, the last two values of C in column C while you group by A alone. Use default output names.

```
# Create a new data.table called DT
set.seed(1L)
DT <- data.table(A = rep(letters[2:1], each = 4L), B = rep(1:4, each = 2L),
                 C = sample(8))
DT
```

```
##    A B C
## 1: b 1 3
## 2: b 1 8
## 3: b 2 4
## 4: b 2 5
## 5: a 3 1
## 6: a 3 7
## 7: a 4 2
## 8: a 4 6
```

```
# DT2
DT2 <- DT[, .(C = cumsum(C)), by = .(A, B)]
# Note: cumsum(x) returns a vector whose elements are the cumulative sums of
# the elements of the argument

# Now the last two values of C from DT2 while you group by A
DT2[, .(C = tail(C, 2)), by = A]
```

```
##    A C
## 1: b 4
## 2: b 9
## 3: a 2
## 4: a 8
```

Chapter 2

Exercise 1 - Create and Subset a data.table

Remember that chaining allows concatenating multiple operations and follows because the operations are in the same order. Furthermore, it helps to avoid the creation of unnecessary temporary variables (which could quickly clutter one's work space).

Instructions

- In the Chapter 1, Exercise 6, we calculated DT2 by taking the cumulative sum of C while grouping by A,B. Next, you selected from DT2 the last two values of C while grouping by A alone. Do this again, but this time use the concept of chaining instead.

```
# DT
set.seed(1L)
DT <- data.table(A = rep(letters[2:1], each = 4L), B = rep(1:4, each = 2L),
                 C = sample(8))

# Previous exercise:
# DT2 <- DT[, .(C = cumsum(C)), by = .(A, B)]
# DT2[, .(C = tail(C, 2)), by = A]
# Now use chaining
DT[,.(C = cumsum(C)), by = .(A, B)][, .(C = tail(C,2)), by = A]
```

```
##    A C
## 1: b 4
## 2: b 9
## 3: a 2
## 4: a 8
```

Exercise 2 - Chaining the iris dataset

Once the **iris** data set is converted to a data.table called DT. Let us see how you can use chaining to simplify manipulations and calculations with the DT.

Instructions

- Get the median of all the four columns Sepal.Length, Sepal.Width, Petal.Length and Petal.Width while you group by Species. Give them the same names. Next, order Species in descending order using chaining.

NOTE: This is deliberately difficult and a little bit repetitive. Also errors can easily be made, especially if there are a large amount of columns in the data set. See the next exercise for a simpler solution.

```r
# load the iris dataset
data(iris)

# iris as data.table
DT <- as.data.table(iris)

# iris
DT[, .(Sepal.Length = median(Sepal.Length), Sepal.Width = median(Sepal.Width),
       Petal.Length = median(Petal.Length), Petal.Width = median(Petal.Width)),
   by = Species][order(-Species)]
```

```
##       Species Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1:  virginica          6.5         3.0         5.55         2.0
## 2: versicolor          5.9         2.8         4.35         1.3
## 3:     setosa          5.0         3.4         1.50         0.2
```

```r
# Note the "-" before Species to denote descending order.
```

Exercise 3 - Progamming time vs. readability

Remember the data.table package provides a special in-built variable **.SD**. It refers to the subset of data for each group. Additionally, it is a good idea to make use of familiar functions from base R to reduce programming time without losing readability.

Instructions

- Get the mean of columns y and z grouped by x by using .SD.
- Get the median of columns y and z grouped by x by using .SD.

```r
# Create a sample data.table
DT <- data.table(x = c(2, 1, 2, 1, 2, 2, 1), y = c(1, 3, 5, 7, 9, 11, 13),
                 z = c(2, 4, 6, 8, 10, 12, 14))
DT
```

```
##    x  y  z
## 1: 2  1  2
## 2: 1  3  4
## 3: 2  5  6
## 4: 1  7  8
## 5: 2  9 10
## 6: 2 11 12
## 7: 1 13 14
```

```
# Mean of columns
DT[, lapply(.SD, mean), by = x]
```

```
##    x        y        z
## 1: 2 6.500000 7.500000
## 2: 1 7.666667 8.666667
```

```
# Median of columns
DT[, lapply(.SD, median), by = x]
```

```
##    x y z
## 1: 2 7 8
## 2: 1 7 8
```

Exercise 4 - Introducing .SDcols

.SDcols specifies the columns of a **data.table** that are included in **.SD**. Using **.SDcols** comes in handy if you have too many columns and you want to perform a particular operation on a particular subset of the columns (instead from the grouping variable columns). Look at **?data.table** for more info on **.SDcols**.

Instructions

- Calculate the sum of the Q columns using .SD and .SDcols. Use 2:4.
- You can of course set .SDcols to be the result of a function call. This time calculate the sum of columns H1 and H2 using paste0().
- Finally, select all but the first row of groups 1 and 2, returning only the grp column and the Q columns. Use -1 in i of .SD and use paste0() again. Type desired_result to better understand what you need to do.

```
# Create a sample data.table
DT <- data.table(grp = c(6, 6, 8, 8, 8), Q1 = c(2, 2, 3, 5, 2),
                 Q2 = c(5, 5, 4, 4, 1), Q3 = c(2, 1, 4, 2, 4),
                 H1 = c(3, 4, 5, 2, 4), H2 = c(5, 2, 4, 1, 2))
DT
```

```
##    grp Q1 Q2 Q3 H1 H2
## 1:   6  2  5  2  3  5
## 2:   6  2  5  1  4  2
## 3:   8  3  4  4  5  4
## 4:   8  5  4  2  2  1
## 5:   8  2  1  4  4  2
```

```
# Calculate the sum of the Q columns
DT[, lapply(.SD, sum), .SDcols = 2:4]
```

```
##    Q1 Q2 Q3
## 1: 14 19 13
```

```
# Calculate the sum of columns H1 and H2
DT[, lapply(.SD, sum), .SDcols = paste0("H", 1:2)]
```

```
##    H1 H2
## 1: 18 14
```

```
# Note: paste() converts its arguments to character strings and concatenates
# them (separating them by the string given by sep). paste0() is equivalent to
# paste(..., sep = "", collapse),

# Select all but the first row of groups 1 and 2, returning only the grp column and the Q columns.
DT[, .SD[-1], by = grp, .SDcols = paste0("Q", 1:3)]
```

```
##    grp Q1 Q2 Q3
## 1:   6  2  5  1
## 2:   8  5  4  2
## 3:   8  2  1  4
```

Exercise 5 - Putting it together: lapply, .SD, .SDcols, .N

Remember that **lapply()** and **.SD** return a list, and that **.N** is an inbuilt variable that returns an integer vector of length 1. If **j** returns a list then a data.table is returned. To combine items together you already know to use c(). Combining a list with a vector makes a new list one longer. When you select **.N** on its own, it automatically gets named N in the output for convenience when chaining.

Instructions

- Get the sum of all columns x, y and z and the number of rows in each group while grouping by x. Your answer should be identical to the result stored in desired_result_1.
- Get the cumulative sum of column x and y while grouping by x and z > 8 such that the answer looks like the result that was stored in the variable desired_result_2.
- Use chaining to get the maximum of both x and y by combining the result that was obtained in the previous instruction and group by by1.

```
# Manaually pre-load the sample data
DT <- data.table(x = c(2, 1, 2, 1, 2, 2, 1), y = c(1, 3, 5, 7, 9, 11, 13),
                 z = c(2, 4, 6, 8, 10, 12, 14))
DT
```

```
##    x  y  z
## 1: 2  1  2
## 2: 1  3  4
## 3: 2  5  6
## 4: 1  7  8
## 5: 2  9 10
## 6: 2 11 12
## 7: 1 13 14
```

```r
# Sum of all columns and the number of rows
# desired_result_1
#    x x  y  z N
#1: 2 8 26 30 4
#2: 1 3 23 26 3
DT[, c(lapply(.SD, sum), .N), by = x, .SDcols = c("x", "y", "z")]
```

```
##    x x  y  z N
## 1: 2 8 26 30 4
## 2: 1 3 23 26 3
```

```r
# Cumulative sum of column `x` and `y` while grouping by `x` and `z > 8`
# desired_result_2
#    by1   by2 x  y
#1:   2 FALSE 2  1
#2:   2 FALSE 4  6
#3:   1 FALSE 1  3
#4:   1 FALSE 2 10
#5:   2  TRUE 2  9
#6:   2  TRUE 4 20
#7:   1  TRUE 1 13
DT[, lapply(.SD, cumsum), by = .(by1 = x, by2 = z > 8), .SDcols = c("x", "y")]
```

```
##    by1   by2 x  y
## 1:   2 FALSE 2  1
## 2:   2 FALSE 4  6
## 3:   1 FALSE 1  3
## 4:   1 FALSE 2 10
## 5:   2  TRUE 2  9
## 6:   2  TRUE 4 20
## 7:   1  TRUE 1 13
```

```r
# Chaining
DT[, lapply(.SD, cumsum), by = .(by1 = x, by2 = z > 8), .SDcols = 1:2][, lapply(.SD, max), by = by1, .S
```

```
##    by1 x  y
## 1:   2 4 20
## 2:   1 2 13
```

Exercise 6 - Adding, Updating and removing columns (LHS := RHS)

Remember that := is defined for use in **j** only, and there are two ways of using it.
The first is the **LHS := RHS** form, where **LHS** is a character vector of column names and **RHS** is the corresponding value.

Instructions

- Add a column Total by reference containing sum(B) for each group in column A.
- Add 1 to column B just in rows 2 and 4.
- Add a new column Total2 that contains sum(B) grouped by A but just over rows 2,3 and 4.

- Remove the Total column. The R function [[is useful to select a single column by name or number with the result returned as a vector. Use [[to select the third column.

NOTE: that for the second instruction in j the performance goes up if you coerce RHS to integer yourself via 1L or via as.integer().

```r
# Manaually pre-load the sample data
DT <- data.table(A = c("a", "a", "a", "b", "b"), B = 1:5)
DT
```

```
##    A B
## 1: a 1
## 2: a 2
## 3: a 3
## 4: b 4
## 5: b 5
```

```r
# Column `Total` by reference
DT[, Total := sum(B), by = A]
```

```
##    A B Total
## 1: a 1     6
## 2: a 2     6
## 3: a 3     6
## 4: b 4     9
## 5: b 5     9
```

```r
DT
```

```
##    A B Total
## 1: a 1     6
## 2: a 2     6
## 3: a 3     6
## 4: b 4     9
## 5: b 5     9
```

```r
# Add 1 to column `B`
DT[c(2, 4), B := B + 1L]
```

```
##    A B Total
## 1: a 1     6
## 2: a 3     6
## 3: a 3     6
## 4: b 5     9
## 5: b 5     9
```

```r
DT
```

```
##    A B Total
## 1: a 1     6
## 2: a 3     6
## 3: a 3     6
## 4: b 5     9
## 5: b 5     9
```

```r
# Add a new column `Total2`
DT[2:4, Total2 := sum(B), by = A]
```

```
##    A B Total Total2
## 1: a 1     6     NA
## 2: a 3     6      6
## 3: a 3     6      6
## 4: b 5     9      5
## 5: b 5     9     NA
```

```r
DT
```

```
##    A B Total Total2
## 1: a 1     6     NA
## 2: a 3     6      6
## 3: a 3     6      6
## 4: b 5     9      5
## 5: b 5     9     NA
```

```r
# Remove the `Total` column
DT[, Total := NULL]
```

```
##    A B Total2
## 1: a 1     NA
## 2: a 3      6
## 3: a 3      6
## 4: b 5      5
## 5: b 5     NA
```

```r
DT
```

```
##    A B Total2
## 1: a 1     NA
## 2: a 3      6
## 3: a 3      6
## 4: b 5      5
## 5: b 5     NA
```

```r
# Select the third column using `[[`
DT[[3]]
```

```
## [1] NA  6  6  5 NA
```

Exercise 7 - The Function form of :=

The second way to use := is a function form. The nice thing about a function form is that you can get each right hand side (RHS) alongside the left hand side (LHS) so it is easier to read.

Instructions

- Update B with B+1 and add a new column C with A+B, and D with constant 2.
- Given myCols = c(“B”,“C”), delete those columns.
- Delete column D by its number (2) rather than its name (D).

```r
# Manaually pre-load the sample data
DT <- data.table(A=c(1,1,1,2,2), B=1:5)
DT
```

```
##    A B
## 1: 1 1
## 2: 1 2
## 3: 1 3
## 4: 2 4
## 5: 2 5
```

```r
# Update `B`, add `C` and `D`
DT[, `:=` (B = B + 1, C = A + B, D = 2)]
```

```
##    A B C D
## 1: 1 2 2 2
## 2: 1 3 3 2
## 3: 1 4 4 2
## 4: 2 5 6 2
## 5: 2 6 7 2
```

```r
DT
```

```
##    A B C D
## 1: 1 2 2 2
## 2: 1 3 3 2
## 3: 1 4 4 2
## 4: 2 5 6 2
## 5: 2 6 7 2
```

```r
# Delete `myCols`
myCols = c("B","C")
DT[, (myCols) := NULL]
```

```
##    A D
## 1: 1 2
## 2: 1 2
## 3: 1 2
## 4: 2 2
## 5: 2 2
```

```r
DT
```

```
##    A D
## 1: 1 2
## 2: 1 2
## 3: 1 2
## 4: 2 2
## 5: 2 2
```

```
# Delete column 2 by number
DT[, 2 := NULL]
```

```
##    A
## 1: 1
## 2: 1
## 3: 1
## 4: 2
## 5: 2
```

```
DT
```

```
##    A
## 1: 1
## 2: 1
## 3: 1
## 4: 2
## 5: 2
```

Exercise 8 - Using set().

Remember that **set()** can not do grouped operations.

Instructions

- Loop through columns 2,3 and 4, and for each one select 3 rows at random and set the value of that column to NA.
- Change the column names to lower case. When setnames() is passed just one input vector, that needs to be all the new names.

```
set.seed(1)

# Manually create DT
DT <- data.table(A = c(2, 2, 3, 5, 2, 5, 5, 4, 4, 1),
                 B = c(2, 1, 4, 2, 4, 3, 4, 5, 2, 4),
                 C = c(5, 2, 4, 1, 2, 2, 1, 2, 5, 2),
                 D = c(3, 3, 3, 1, 5, 4, 4, 1, 4, 3))
DT
```

```
##     A B C D
##  1: 2 2 5 3
##  2: 2 1 2 3
##  3: 3 4 4 3
##  4: 5 2 1 1
##  5: 2 4 2 5
##  6: 5 3 2 4
##  7: 5 4 1 4
##  8: 4 5 2 1
##  9: 4 2 5 4
## 10: 1 4 2 3
```

```
# For loop with set
for(i in 2:4)
    set(DT, sample(10, 3), i, NA)

# Change the column names to lowercase
setnames(DT, tolower(names(DT)))

# Print the new DT to the console to confirm changes
DT
```

```
##     a  b  c  d
##  1: 2  2  5  3
##  2: 2  1 NA  3
##  3: 3 NA  4  3
##  4: 5 NA  1  1
##  5: 2 NA  2  5
##  6: 5  3  2 NA
##  7: 5  4  1  4
##  8: 4  5 NA  1
##  9: 4  2  5 NA
## 10: 1  4 NA NA
```

Exercise 9 - The set() family

Remember the following:
1. set() is a loopable low overhead version of ":="
2. You can use setnames() to set or change column names
3. setcolorder() lets you reorder the columns of a data.table

Instructions

- Add a postfix "_2" to all column names.
- Change "a_2" to "A2" by names not position.
- To end, reverse the order of the columns.

```
# Create the sample dataset
DT <- data.table(a=letters[c(1,1,1,2,2)], b=1)
DT
```

```
##    a b
## 1: a 1
## 2: a 1
## 3: a 1
## 4: b 1
## 5: b 1
```

```r
# Postfix "_2"
setnames(DT, paste0(names(DT), "_2"))
DT
```

```
##    a_2 b_2
## 1:   a   1
## 2:   a   1
## 3:   a   1
## 4:   b   1
## 5:   b   1
```

```r
# "a_2" to "A2"
setnames(DT, "a_2", "A2")
DT
```

```
##    A2 b_2
## 1:  a   1
## 2:  a   1
## 3:  a   1
## 4:  b   1
## 5:  b   1
```

```r
# Reversing order
# NOTE: the usage of the 2:1
setcolorder(DT, 2:1)
DT
```

```
##    b_2 A2
## 1:   1  a
## 2:   1  a
## 3:   1  a
## 4:   1  b
## 5:   1  b
```

Chapter 3

Exercise 1 - Selecting rows the data.table way

Remember that you can use column names in **i** to select certain rows. Since selecting rows over and over again is common place with **data.table**, it is best to do a small exercise on this with the help of the familiar iris data set.

NOTE: This is different from selecting columns and running functions on columns using **j**.

Instructions

- Convert the iris data set to a data.table with the same name iris
- Select all the rows where Species is “virginica”
- Select all the rows where Species is either “virginica” or “versicolor”

```r
# iris as data.table
iris <- as.data.table(iris)

# Species is "virginica"
iris[Species == "virginica"]
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
##  1:          6.3         3.3          6.0         2.5 virginica
##  2:          5.8         2.7          5.1         1.9 virginica
##  3:          7.1         3.0          5.9         2.1 virginica
##  4:          6.3         2.9          5.6         1.8 virginica
##  5:          6.5         3.0          5.8         2.2 virginica
##  6:          7.6         3.0          6.6         2.1 virginica
##  7:          4.9         2.5          4.5         1.7 virginica
##  8:          7.3         2.9          6.3         1.8 virginica
##  9:          6.7         2.5          5.8         1.8 virginica
## 10:          7.2         3.6          6.1         2.5 virginica
## 11:          6.5         3.2          5.1         2.0 virginica
## 12:          6.4         2.7          5.3         1.9 virginica
## 13:          6.8         3.0          5.5         2.1 virginica
## 14:          5.7         2.5          5.0         2.0 virginica
## 15:          5.8         2.8          5.1         2.4 virginica
## 16:          6.4         3.2          5.3         2.3 virginica
## 17:          6.5         3.0          5.5         1.8 virginica
## 18:          7.7         3.8          6.7         2.2 virginica
## 19:          7.7         2.6          6.9         2.3 virginica
## 20:          6.0         2.2          5.0         1.5 virginica
## 21:          6.9         3.2          5.7         2.3 virginica
## 22:          5.6         2.8          4.9         2.0 virginica
## 23:          7.7         2.8          6.7         2.0 virginica
## 24:          6.3         2.7          4.9         1.8 virginica
## 25:          6.7         3.3          5.7         2.1 virginica
## 26:          7.2         3.2          6.0         1.8 virginica
## 27:          6.2         2.8          4.8         1.8 virginica
## 28:          6.1         3.0          4.9         1.8 virginica
## 29:          6.4         2.8          5.6         2.1 virginica
## 30:          7.2         3.0          5.8         1.6 virginica
## 31:          7.4         2.8          6.1         1.9 virginica
## 32:          7.9         3.8          6.4         2.0 virginica
## 33:          6.4         2.8          5.6         2.2 virginica
## 34:          6.3         2.8          5.1         1.5 virginica
## 35:          6.1         2.6          5.6         1.4 virginica
## 36:          7.7         3.0          6.1         2.3 virginica
## 37:          6.3         3.4          5.6         2.4 virginica
## 38:          6.4         3.1          5.5         1.8 virginica
## 39:          6.0         3.0          4.8         1.8 virginica
## 40:          6.9         3.1          5.4         2.1 virginica
## 41:          6.7         3.1          5.6         2.4 virginica
## 42:          6.9         3.1          5.1         2.3 virginica
## 43:          5.8         2.7          5.1         1.9 virginica
## 44:          6.8         3.2          5.9         2.3 virginica
## 45:          6.7         3.3          5.7         2.5 virginica
## 46:          6.7         3.0          5.2         2.3 virginica
## 47:          6.3         2.5          5.0         1.9 virginica
## 48:          6.5         3.0          5.2         2.0 virginica
## 49:          6.2         3.4          5.4         2.3 virginica
## 50:          5.9         3.0          5.1         1.8 virginica
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
```

```
# Species is either "virginica" or "versicolor"
iris[Species %in% c("virginica", "versicolor")]
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
##   1:          7.0         3.2          4.7         1.4 versicolor
##   2:          6.4         3.2          4.5         1.5 versicolor
##   3:          6.9         3.1          4.9         1.5 versicolor
##   4:          5.5         2.3          4.0         1.3 versicolor
##   5:          6.5         2.8          4.6         1.5 versicolor
##   6:          5.7         2.8          4.5         1.3 versicolor
##   7:          6.3         3.3          4.7         1.6 versicolor
##   8:          4.9         2.4          3.3         1.0 versicolor
##   9:          6.6         2.9          4.6         1.3 versicolor
##  10:          5.2         2.7          3.9         1.4 versicolor
##  11:          5.0         2.0          3.5         1.0 versicolor
##  12:          5.9         3.0          4.2         1.5 versicolor
##  13:          6.0         2.2          4.0         1.0 versicolor
##  14:          6.1         2.9          4.7         1.4 versicolor
##  15:          5.6         2.9          3.6         1.3 versicolor
##  16:          6.7         3.1          4.4         1.4 versicolor
##  17:          5.6         3.0          4.5         1.5 versicolor
##  18:          5.8         2.7          4.1         1.0 versicolor
##  19:          6.2         2.2          4.5         1.5 versicolor
##  20:          5.6         2.5          3.9         1.1 versicolor
##  21:          5.9         3.2          4.8         1.8 versicolor
##  22:          6.1         2.8          4.0         1.3 versicolor
##  23:          6.3         2.5          4.9         1.5 versicolor
##  24:          6.1         2.8          4.7         1.2 versicolor
##  25:          6.4         2.9          4.3         1.3 versicolor
##  26:          6.6         3.0          4.4         1.4 versicolor
##  27:          6.8         2.8          4.8         1.4 versicolor
##  28:          6.7         3.0          5.0         1.7 versicolor
##  29:          6.0         2.9          4.5         1.5 versicolor
##  30:          5.7         2.6          3.5         1.0 versicolor
##  31:          5.5         2.4          3.8         1.1 versicolor
##  32:          5.5         2.4          3.7         1.0 versicolor
##  33:          5.8         2.7          3.9         1.2 versicolor
##  34:          6.0         2.7          5.1         1.6 versicolor
##  35:          5.4         3.0          4.5         1.5 versicolor
##  36:          6.0         3.4          4.5         1.6 versicolor
##  37:          6.7         3.1          4.7         1.5 versicolor
##  38:          6.3         2.3          4.4         1.3 versicolor
##  39:          5.6         3.0          4.1         1.3 versicolor
##  40:          5.5         2.5          4.0         1.3 versicolor
##  41:          5.5         2.6          4.4         1.2 versicolor
##  42:          6.1         3.0          4.6         1.4 versicolor
##  43:          5.8         2.6          4.0         1.2 versicolor
##  44:          5.0         2.3          3.3         1.0 versicolor
## 45:          5.6         2.7          4.2         1.3 versicolor
## 46:          5.7         3.0          4.2         1.2 versicolor
## 47:          5.7         2.9          4.2         1.3 versicolor
## 48:          6.2         2.9          4.3         1.3 versicolor
## 49:          5.1         2.5          3.0         1.1 versicolor
## 50:          5.7         2.8          4.1         1.3 versicolor
## 51:          6.3         3.3          6.0         2.5  virginica
## 52:          5.8         2.7          5.1         1.9  virginica
## 53:          7.1         3.0          5.9         2.1  virginica
## 54:          6.3         2.9          5.6         1.8  virginica
## 55:          6.5         3.0          5.8         2.2  virginica
## 56:          7.6         3.0          6.6         2.1  virginica
## 57:          4.9         2.5          4.5         1.7  virginica
## 58:          7.3         2.9          6.3         1.8  virginica
## 59:          6.7         2.5          5.8         1.8  virginica
## 60:          7.2         3.6          6.1         2.5  virginica
## 61:          6.5         3.2          5.1         2.0  virginica
## 62:          6.4         2.7          5.3         1.9  virginica
## 63:          6.8         3.0          5.5         2.1  virginica
## 64:          5.7         2.5          5.0         2.0  virginica
## 65:          5.8         2.8          5.1         2.4  virginica
## 66:          6.4         3.2          5.3         2.3  virginica
## 67:          6.5         3.0          5.5         1.8  virginica
## 68:          7.7         3.8          6.7         2.2  virginica
## 69:          7.7         2.6          6.9         2.3  virginica
## 70:          6.0         2.2          5.0         1.5  virginica
## 71:          6.9         3.2          5.7         2.3  virginica
## 72:          5.6         2.8          4.9         2.0  virginica
## 73:          7.7         2.8          6.7         2.0  virginica
## 74:          6.3         2.7          4.9         1.8  virginica
## 75:          6.7         3.3          5.7         2.1  virginica
## 76:          7.2         3.2          6.0         1.8  virginica
## 77:          6.2         2.8          4.8         1.8  virginica
## 78:          6.1         3.0          4.9         1.8  virginica
## 79:          6.4         2.8          5.6         2.1  virginica
## 80:          7.2         3.0          5.8         1.6  virginica
## 81:          7.4         2.8          6.1         1.9  virginica
## 82:          7.9         3.8          6.4         2.0  virginica
## 83:          6.4         2.8          5.6         2.2  virginica
## 84:          6.3         2.8          5.1         1.5  virginica
## 85:          6.1         2.6          5.6         1.4  virginica
## 86:          7.7         3.0          6.1         2.3  virginica
## 87:          6.3         3.4          5.6         2.4  virginica
## 88:          6.4         3.1          5.5         1.8  virginica
## 89:          6.0         3.0          4.8         1.8  virginica
## 90:          6.9         3.1          5.4         2.1  virginica
## 91:          6.7         3.1          5.6         2.4  virginica
## 92:          6.9         3.1          5.1         2.3  virginica
## 93:          5.8         2.7          5.1         1.9  virginica
## 94:          6.8         3.2          5.9         2.3  virginica
## 95:          6.7         3.3          5.7         2.5  virginica
## 96:          6.7         3.0          5.2         2.3  virginica
## 97:          6.3         2.5          5.0         1.9  virginica
## 98:          6.5         3.0          5.2         2.0  virginica
##  99:          6.2         3.4          5.4         2.3  virginica
## 100:          5.9         3.0          5.1         1.8  virginica
##      Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
```

Exercise 2 - Removing columns and adapting column names

In the previous exercise you needed to select certain rows out of the iris data.table based on the column names. Now you have to take your understanding of the data.table package to the next level: use standard R functions and regular expressions to manipulate your data.table by removing columns and adapting column names. To practice this, we'll do a little manipulation to prepare easier column names for the next exercise.

Instructions

- Remove the "Sepal." prefix and remove the two Petal columns from the iris data.table.

```
# Convert iris to a data.table
iris <- as.data.table(iris)

# Remove the "Sepal." prefix...
# Note that gsub() performs replacement of all matches
# of the regular expression
setnames(iris, gsub("^Sepal[.]", "", names(iris)))
iris
```

```
##      Length Width Petal.Length Petal.Width   Species
##   1:    5.1   3.5          1.4         0.2    setosa
##   2:    4.9   3.0          1.4         0.2    setosa
##   3:    4.7   3.2          1.3         0.2    setosa
##   4:    4.6   3.1          1.5         0.2    setosa
##   5:    5.0   3.6          1.4         0.2    setosa
##  ---
## 146:    6.7   3.0          5.2         2.3 virginica
## 147:    6.3   2.5          5.0         1.9 virginica
## 148:    6.5   3.0          5.2         2.0 virginica
## 149:    6.2   3.4          5.4         2.3 virginica
## 150:    5.9   3.0          5.1         1.8 virginica
```

```
# ...and remove the two `Petal` columns
iris[, grep("^Petal", names(iris)) := NULL]
```

```
##      Length Width   Species
##   1:    5.1   3.5    setosa
##   2:    4.9   3.0    setosa
##   3:    4.7   3.2    setosa
##   4:    4.6   3.1    setosa
##   5:    5.0   3.6    setosa
##  ---
## 146:    6.7   3.0 virginica
## 147:    6.3   2.5 virginica
## 148:    6.5   3.0 virginica
## 149:    6.2   3.4 virginica
## 150:    5.9   3.0 virginica
```

```
iris
```

```
##      Length Width   Species
##   1:    5.1   3.5    setosa
##   2:    4.9   3.0    setosa
##   3:    4.7   3.2    setosa
##   4:    4.6   3.1    setosa
##   5:    5.0   3.6    setosa
##  ---
## 146:    6.7   3.0 virginica
## 147:    6.3   2.5 virginica
## 148:    6.5   3.0 virginica
## 149:    6.2   3.4 virginica
## 150:    5.9   3.0 virginica
```

Exercise 3 - Understanding automatic indexing

Remeber the rule that **"if i is a single variable name, it is evaluated in calling scope, otherwise inside DT's scope"**. This is a very important rule if you want to conceptually understand what is going on when using column names in **i**. Only single columns on the left of operators benefit from automatic indexing.

Instructions

- Select the rows where the area is greater than 20 square centimeters.
- Add a new boolean column containing Width x Length > 20 and call it IsLarge.
- Select the rows where IsLarge is TRUE.

```
# Use the data.table iris that is already loaded
iris
```

```
##      Length Width   Species
##   1:    5.1   3.5    setosa
##   2:    4.9   3.0    setosa
##   3:    4.7   3.2    setosa
##   4:    4.6   3.1    setosa
##   5:    5.0   3.6    setosa
##  ---
## 146:    6.7   3.0 virginica
## 147:    6.3   2.5 virginica
## 148:    6.5   3.0 virginica
## 149:    6.2   3.4 virginica
## 150:    5.9   3.0 virginica
```

```
# Area is greater than 20 square centimeters
iris[Width * Length > 20]
```

```
##     Length Width    Species
##  1:    5.4   3.9     setosa
##  2:    5.8   4.0     setosa
##  3:    5.7   4.4     setosa
##  4:    5.4   3.9     setosa
##  5:    5.7   3.8     setosa
##  6:    5.2   4.1     setosa
##  7:    5.5   4.2     setosa
##  8:    7.0   3.2 versicolor
##  9:    6.4   3.2 versicolor
## 10:    6.9   3.1 versicolor
## 11:    6.3   3.3 versicolor
## 12:    6.7   3.1 versicolor
## 13:    6.7   3.0 versicolor
## 14:    6.0   3.4 versicolor
## 15:    6.7   3.1 versicolor
## 16:    6.3   3.3  virginica
## 17:    7.1   3.0  virginica
## 18:    7.6   3.0  virginica
## 19:    7.3   2.9  virginica
## 20:    7.2   3.6  virginica
## 21:    6.5   3.2  virginica
## 22:    6.8   3.0  virginica
## 23:    6.4   3.2  virginica
## 24:    7.7   3.8  virginica
## 25:    7.7   2.6  virginica
## 26:    6.9   3.2  virginica
## 27:    7.7   2.8  virginica
## 28:    6.7   3.3  virginica
## 29:    7.2   3.2  virginica
## 30:    7.2   3.0  virginica
## 31:    7.4   2.8  virginica
## 32:    7.9   3.8  virginica
## 33:    7.7   3.0  virginica
## 34:    6.3   3.4  virginica
## 35:    6.9   3.1  virginica
## 36:    6.7   3.1  virginica
## 37:    6.9   3.1  virginica
## 38:    6.8   3.2  virginica
## 39:    6.7   3.3  virginica
## 40:    6.7   3.0  virginica
## 41:    6.2   3.4  virginica
##     Length Width    Species
```

```
# Now demonstrate the above conceptually
# Add new boolean column
iris[, IsLarge := Width * Length > 20]
```

```
##      Length Width    Species IsLarge
##   1:    5.1   3.5     setosa   FALSE
##   2:    4.9   3.0     setosa   FALSE
##   3:    4.7   3.2     setosa   FALSE
##   4:    4.6   3.1     setosa   FALSE
##   5:    5.0   3.6     setosa   FALSE
##  ---
## 146:    6.7   3.0  virginica    TRUE
## 147:    6.3   2.5  virginica   FALSE
## 148:    6.5   3.0  virginica   FALSE
## 149:    6.2   3.4  virginica    TRUE
## 150:    5.9   3.0 virginica   FALSE
```

```
iris
```

```
##      Length Width   Species IsLarge
##   1:    5.1   3.5    setosa   FALSE
##   2:    4.9   3.0    setosa   FALSE
##   3:    4.7   3.2    setosa   FALSE
##   4:    4.6   3.1    setosa   FALSE
##   5:    5.0   3.6    setosa   FALSE
##  ---
## 146:    6.7   3.0 virginica    TRUE
## 147:    6.3   2.5 virginica   FALSE
## 148:    6.5   3.0 virginica   FALSE
## 149:    6.2   3.4 virginica    TRUE
## 150:    5.9   3.0 virginica   FALSE
```

```
# Now select rows again where the area is greater than 20 square centimeters
iris[IsLarge == TRUE]
```

```
##     Length Width    Species IsLarge
##  1:    5.4   3.9     setosa    TRUE
##  2:    5.8   4.0     setosa    TRUE
##  3:    5.7   4.4     setosa    TRUE
##  4:    5.4   3.9     setosa    TRUE
##  5:    5.7   3.8     setosa    TRUE
##  6:    5.2   4.1     setosa    TRUE
##  7:    5.5   4.2     setosa    TRUE
##  8:    7.0   3.2 versicolor    TRUE
##  9:    6.4   3.2 versicolor    TRUE
## 10:    6.9   3.1 versicolor    TRUE
## 11:    6.3   3.3 versicolor    TRUE
## 12:    6.7   3.1 versicolor    TRUE
## 13:    6.7   3.0 versicolor    TRUE
## 14:    6.0   3.4 versicolor    TRUE
## 15:    6.7   3.1 versicolor    TRUE
## 16:    6.3   3.3  virginica    TRUE
## 17:    7.1   3.0  virginica    TRUE
## 18:    7.6   3.0  virginica    TRUE
## 19:    7.3   2.9  virginica    TRUE
## 20:    7.2   3.6  virginica    TRUE
## 21:    6.5   3.2  virginica    TRUE
## 22:    6.8   3.0  virginica    TRUE
## 23:    6.4   3.2  virginica    TRUE
## 24:    7.7   3.8  virginica    TRUE
## 25:    7.7   2.6  virginica    TRUE
## 26:    6.9   3.2  virginica    TRUE
## 27:    7.7   2.8  virginica    TRUE
## 28:    6.7   3.3  virginica    TRUE
## 29:    7.2   3.2  virginica    TRUE
## 30:    7.2   3.0  virginica    TRUE
## 31:    7.4   2.8  virginica    TRUE
## 32:    7.9   3.8  virginica    TRUE
## 33:    7.7   3.0  virginica    TRUE
## 34:    6.3   3.4  virginica    TRUE
## 35:    6.9   3.1  virginica    TRUE
## 36:    6.7   3.1  virginica    TRUE
## 37:    6.9   3.1  virginica    TRUE
## 38:    6.8   3.2  virginica    TRUE
## 39:    6.7   3.3  virginica    TRUE
## 40:    6.7   3.0  virginica    TRUE
## 41:    6.2   3.4  virginica    TRUE
##     Length Width    Species IsLarge
```

```
# Note that iris[(IsLarge)] can also be used
```

Exercise 4 - Testing the understanding of KEYs

Instructions Load the sample data.table (DT) and execute the following:

```
# Load the sample data.table
DT <- data.table(A = c("b", "a", "b", "c", "a", "b", "c"),
                 B = c(2, 4, 1, 7, 5, 3, 6),
                 C = 6:12)
```

1. Select the b group using ==.

```
DT[A == "b"]
```

```
##    A B  C
## 1: b 2  6
## 2: b 1  8
## 3: b 3 11
```

2. Set a 2 column key on A and B.

```
setkey(DT, A, B)
```

3. Select the b group again: again use ==.

```
DT[A == "b"]
```

```
##    A B  C
## 1: b 1  8
## 2: b 2  6
## 3: b 3 11
```

Note that the order of the rows within the b group changed, because **B** is included in the key.

Exercise 5 - Selecting groups or parts of groups

The previous exercise illustrated how you can manually set a key via **setkey(DT, A, B)**. **setkey()** will then sort the data by the columns that you specify, and change the table by reference. Having set a key will allow you to use it as a **super-charged** rowname when doing selections for example. Arguments like **mult** and **nomatch** then help you to only select particular parts of groups.

Furthermore, two of the instructions will require you to make use of **by=.EACHI**. This will allow you to run **j** for each group in which each item in **i** joins too.

Instructions

- Select the b group without using ==.
- Select the b and c groups.
- Select the first row of the b and c groups using mult.
- Use by=.EACHI and .SD to select the first and last row of the b and c groups.
- Extend the previous command to print out the group before returning the first and last row from it.

```r
# Create the sample data.table (DT).
# Set the keys to `A` and `B`
DT <- data.table(A = letters[c(2,1,2,3,1,2,3)],
                 B = c(5,4,1,9,8,8,6), C=6:12)
setkey(DT,A,B)

# Select the `b` group
DT["b"]
```

```
##    A B  C
## 1: b 1  8
## 2: b 5  6
## 3: b 8 11
```

```r
# `b` and `c` groups
DT[c("b", "c")]
```

```
##    A B  C
## 1: b 1  8
## 2: b 5  6
## 3: b 8 11
## 4: c 6 12
## 5: c 9  9
```

```r
# The first row of the `b` and `c` group
DT[c("b", "c"), mult = "first"]
```

```
##    A B  C
## 1: b 1  8
## 2: c 6 12
```

```r
# `by=.EACHI` and `.SD`
DT[c("b", "c"), .SD[c(1, .N)], by = .EACHI]
```

```
##    A B  C
## 1: b 1  8
## 2: b 8 11
## 3: c 6 12
## 4: c 9  9
```

```r
# Print out all the data in the two groups before you return the first and last row of each group again
DT[c("b", "c"), {print(.SD); .SD[c(1, .N)]}, by = .EACHI]
```

```
##    B  C
## 1: 1  8
## 2: 5  6
## 3: 8 11
##    B  C
## 1: 6 12
## 2: 9  9
```

```
##    A B  C
## 1: b 1  8
## 2: b 8 11
## 3: c 6 12
## 4: c 9  9
```

Exercise 6 - Rolling joinbs (Part one)

Remember that the roll applies to the last join column.

Instructions

- Get the key of DT via the key() function.
- Use the super charged row names to lookup the row where A=="b" & B==6, without using ==. Verify here that C is NA.
- Base yourself on the query that was written in the previous instruction to return the prevailing row before this gap.
- Similar, but this time find the nearest one.

```r
# Load the samplke the data.table `DT`
DT <- data.table(A = letters[c(1, 1, 2, 2, 2, 3, 3)],
                 B = c(4, 8, 1, 5, 8, 6, 9),
                 C = c(7, 10, 8, 6, 11, 12, 9))
setkey(DT, A, B)

# Key of `DT`
key(DT)
```

```
## [1] "A" "B"
```

```r
# Row where  `A=="b"` & `B==6`
DT[.("b", 6)]
```

```
##    A B  C
## 1: b 6 NA
```

```
# Return the prevailing row
DT[.("b", 6), roll = TRUE]
```

```
##    A B C
## 1: b 6 6
```

```
# Nearest one
DT[.("b", 6), roll = "nearest"]
```

```
##    A B C
## 1: b 6 6
```

Exercise 7 - Rolling joins (Part two)

Remember that the **rollends** argument is actually a vector of two logical values. If you want to roll for a certain distance, you remain with the roll argument.

Instructions

- Look at the sequence (-2):10 for the b group.
- Now carry the prevailing values forwards through the NA.
- Next, carry the first observation backwards as well.
- Finally, only roll for a distance of 2, still with rollends=TRUE. Check that only 4 and -2 now have NA. This is because they are more than 2 from the prevailing observation.

NOTE: The desired result should look as follows:

```
#    A  B  C
# 1: b -2 NA
# 2: b -1 NA
# 3: b  0 NA
# 4: b  1  8
# 5: b  2  8
# 6: b  3  8
# 7: b  4 NA
# 8: b  5  6
# 9: b  6  6
#10: b  7  6
#11: b  8 11
#12: b  9 11
#13: b 10 11
```

```
# Load the samplke the data.table `DT`
DT <- data.table(A = letters[c(1, 1, 2, 2, 2, 3, 3)],
                 B = c(4, 8, 1, 5, 8, 6, 9),
                 C = c(7, 10, 8, 6, 11, 12, 9))
setkey(DT, A, B)

# Look at the sequence (-2):10 for the `b` group
DT[.("b", (-2):10)]
```

```
##      A  B  C
##  1: b -2 NA
##  2: b -1 NA
##  3: b  0 NA
##  4: b  1  8
##  5: b  2 NA
##  6: b  3 NA
##  7: b  4 NA
##  8: b  5  6
##  9: b  6 NA
## 10: b  7 NA
## 11: b  8 11
## 12: b  9 NA
## 13: b 10 NA
```

```
# Carry the prevailing values forwards
DT[.("b", (-2):10), roll = TRUE]
```

```
##      A  B  C
##  1: b -2 NA
##  2: b -1 NA
##  3: b  0 NA
##  4: b  1  8
##  5: b  2  8
##  6: b  3  8
##  7: b  4  8
##  8: b  5  6
##  9: b  6  6
## 10: b  7  6
## 11: b  8 11
## 12: b  9 11
## 13: b 10 11
```

```
# Carry the first observation backwards
DT[.("b", (-2):10), roll = TRUE, rollends = TRUE]
```

```
##      A  B  C
##  1: b -2  8
##  2: b -1  8
##  3: b  0  8
##  4: b  1  8
##  5: b  2  8
##  6: b  3  8
##  7: b  4  8
##  8: b  5  6
##  9: b  6  6
## 10: b  7  6
## 11: b  8 11
## 12: b  9 11
## 13: b 10 11
```

```r
# Note that the following can also be used
# DT[.("b", (-2):10), roll = TRUE, rollends = c(TRUE, TRUE)]

# Roll for a distance of 2
DT[.("b", (-2):10, roll = 2)]
```

```
##      A  B  C roll
##  1: b -2 NA    2
##  2: b -1 NA    2
##  3: b  0 NA    2
##  4: b  1  8    2
##  5: b  2 NA    2
##  6: b  3 NA    2
##  7: b  4 NA    2
##  8: b  5  6    2
##  9: b  6 NA    2
## 10: b  7 NA    2
## 11: b  8 11    2
## 12: b  9 NA    2
## 13: b 10 NA    2
```

Exercise 8 - Rolling joins (Part three)

If you look up the value B==20 in group A=="b" without limiting the roll, the value of column C is now **11**.

```r
DT[.("b", 20), roll = TRUE]
```

```
##    A  B  C
## 1: b 20 11
```