Session - 4

Module - II

- Introduction to *data.table* R package: syntax, usage and benefits
- Merging datasets
- Long form and wide form

- Plotting in R
  - legends, colors, line types, …
  - Multiple lines, multiple axes, multiple plots

data.table

data.table

- It’s a (very) fast, memory efficient and flexible package to analyse data
  - I haven’t used data.frame since discovering data.table
- Extends data.frame
  - Most of the data.frame code will work

data.table

- It’s a (very) fast, memory efficient and flexible package to analyse data
  - I haven’t used data.frame since discovering data.table
- Extends data.frame
  - Most of the data.frame code will work
- Has a “different” and succinct syntax
  - May take some time to learn. But the effort is worth the benefits!
- Have a look:
  - https://cran.r-project.org/web/packages/data.table/index.html
  - github: https://github.com/Rdatatable/data.table

data.table

- It’s a (very) fast, memory efficient and flexible package to analyse data
  - I haven’t used data.frame since discovering data.table
- Extends data.frame
  - Most of the data.frame code will work
- Has a “different” and succinct syntax
  - May take some time to learn. But the effort is worth the benefits!
- Have a look:
  - https://cran.r-project.org/web/packages/data.table/index.html
  - github: https://github.com/Rdatatable/data.table
- Parallelized read and write
  - Very useful while reading GBs of raw data
    - Upto 5x – 10x speedup
    - Speed becomes a big issue when working with huge datasets

Syntax

- Column names can be used as variables
  - `dt[Date > “2020-01-01”]` is valid.
  - Remember in data.frame you need `dt[dt$Date > “2020-01-01”]`

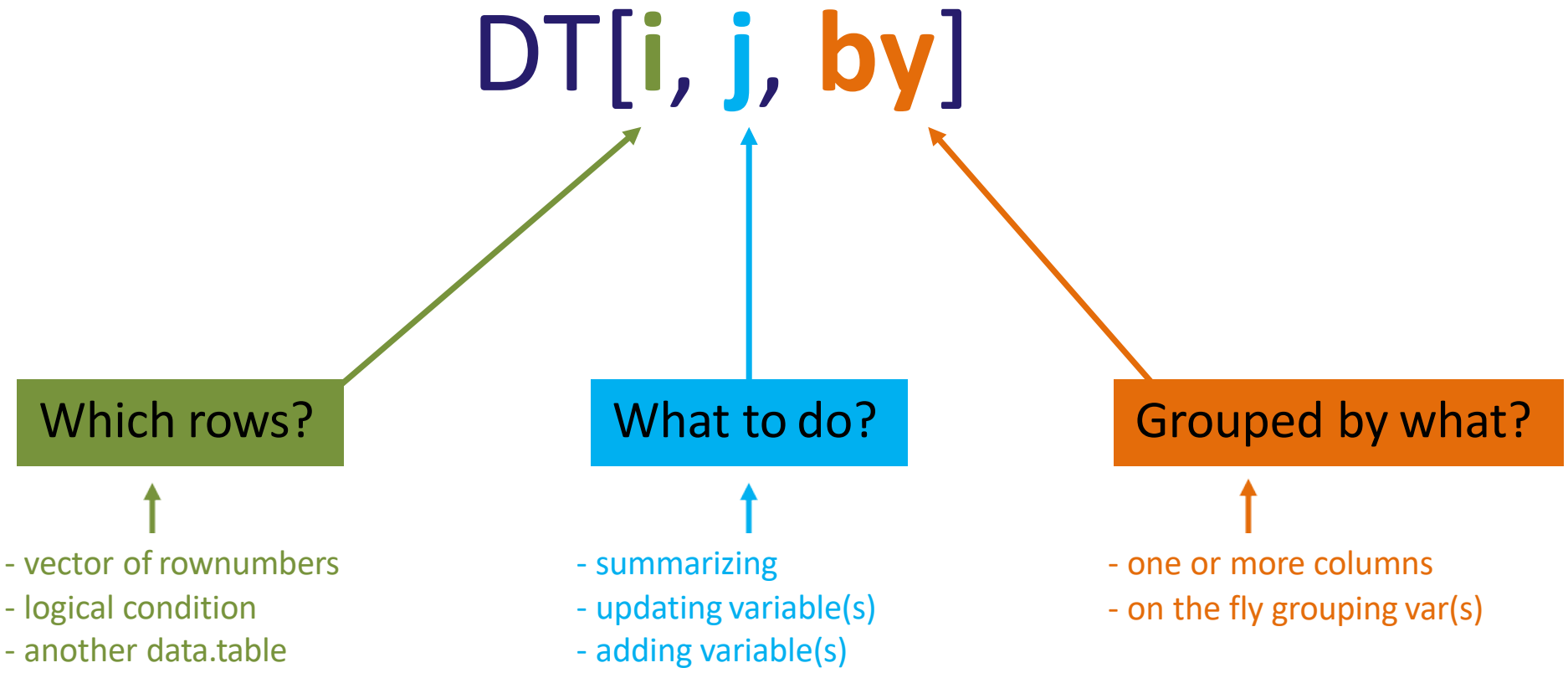
  - Column names are infact variables inside `[]`
  - You can do `dt[, min(Date)]` to get the first date
  - In data.frame you must supply `dt$Date` to min externally.
    - `setDT()` and `setDF()`

- Grouping within `[]` syntax
  - Most of data analysis requires some form of grouping

Syntax: general form

DT[i, j, by]

Syntax: general form



Analogy with SQL (queries)

DT[i, j, by]

```
data.table:     i                   j          by
       SQL: where    select | update    group by
```

Examples

```
subset rows                :        airquality[Day <= 10]
```

```
select columns             :        airquality[, .(Month)]
```

- `airquality` dataset is present in package `datasets`

Examples

```
subset rows                 :       airquality[Day <= 10]

select columns              :       airquality[, .(Month)]

subset rows & select columns  :
                                    airquality[Day %between% c(5, 10), .(Wind, Temp)]
```

- `airquality` dataset is present in package `datasets`

Counts

syntax: DT[**i**, **j**, **by**]

special symbol: **.N**

count

```
iris[Species == "setosa", .N]
```

count distinct

```
iris[, uniqueN(Species)]
iris[Petal.Width < 0.9, uniqueN(Species)]
```

Aggregating

syntax: DT[**i**, **j**, **by**]

Simple aggregation: iris[, .(count = .N, average = mean(Petal.Width))]

Including filtering: iris[Petal.Width < 0.9, .(count = .N, average = mean(Petal.Width))]

Group by

syntax: DT[**i**, **j**, **by**]

```
iris[, .N, by = Species]
```

```
iris[, .(avg = mean(Petal.Width)), by = Species]
```

```
iris[Sepal.Length < 5.3, .(avg = mean(Petal.Width)), by = Species]
```

```
iris[, .(avg = mean(Petal.Width)), by = .(Species, logi = Sepal.Length < 5.3)]
```

Group by

special symbol: **.SD**

SD = **S**ubset of **D**ata

- a data.table by itself
- holds data of current goup as defined in by
- when no by, .SD applies to whole data.table
- allows for calculations on multiple columns

Group by

special symbol: **.SD**

```
iris[, lapply(.SD, mean), by = Species]
```

```
iris[Sepal.Length < 5.3, lapply(.SD, mean), by = Species]
```

Group by

special symbol: **.SD**

special symbol: **.SDcols**

```
iris[, lapply(.SD, mean), by = Species, .SDcols = 1:2]
```

```
iris[, lapply(.SD, mean), by = Species, .SDcols = grep("Length", names(iris))]
```

Examples

syntax: DT[**i**, **j**, **by**]

Count the number of days per month

airquality[, .N, by = Month]

Examples

syntax: DT[**i**, **j**, **by**]

Count the number of days per month

```
airquality[, .N, by = Month]
```

Calculate the average Wind speed by month for only those days that have an ozone value

```
airquality[!is.na(Ozone), mean(Wind), by = Month]
```

Examples

syntax: DT[**i**, **j**, **by**]

Count the number of days per month

```
airquality[, .N, by = Month]
```

Calculate the average Wind speed by month for only those days that have an ozone value

```
airquality[!is.na(Ozone), mean(Wind), by = Month]
```

Calculate the mean temperature for the odd and even days for each month

```
airquality[, mean(Temp)
     , by = .(Month, odd = Day %% 2)]
```

Updating, adding & deleting variables

special operator: **:=**

- updates a data.table in place (by reference)
- can be used to:
  - update existing column(s)
  - add new column(s)
  - delete column(s)

- you don't need **<-** **OR** **=**

Updating variables

special operator: **:=**

```
iris[, Sepal.Length := Sepal.Length *  2]
```

```
iris[, `:=` (Sepal.Length = Sepal.Length * 2,
             Petal.Width = Petal.Width /  2)]
```

Updating variables by group

special operator: **:=**

```
iris[, Sepal.Length := Sepal.Length * uniqueN(Sepal.Width) / .N, by = Species]
```

```
iris[, `:=` (Sepal.Length = Sepal.Length * uniqueN(Sepal.Width),
                Petal.Width = Petal.Width /  .N)
        , by =  Species]
```

Adding variables

special operator: **:=**

special symbol: **.I**

```
iris[, rownumber := .I]
```

```
iris[, Sepal.Area := Sepal.Length *  Sepal.Width]
```

```
iris[, `:=` (Sepal.Area = Sepal.Length * Sepal.Width,
                Petal.Area = Petal.Length *  Petal.Width)]
```

Adding variables by group

special operator: **:=**

```
iris[, Total.Sepal.Area := sum(Sepal.Area), by = Species]
```

```
iris[, `:=` (Total.Sepal.Area = sum(Sepal.Area),
                Total.Petal.Area = sum(Petal.Area))
       , by = Species]
```

Deleting variables

special operator: **:=**

```
iris[, Sepal.Length := NULL]

iris[, (1:4) := NULL]

iris[, grep("Length", names(irisDT)) := NULL]
```

Examples

Change the Wind column from miles per hour to kilometers per hour (1 mph = 1.6 kmh)

```
airquality[, Wind := Wind * 1.6]
```

Examples

Change the Wind column from miles per hour to kilometers per hour (1 mph = 1.6 kmh)

```
airquality[, Wind := Wind * 1.6]
```

Calculate a new **chill** variable (Wind * Temperature)

```
airquality[, chill := Wind * Temp]
```

Examples

Change the Wind column from miles per hour to kilometers per hour (1 mph = 1.6 kmh)

```
airquality[, Wind := Wind * 1.6]
```

Calculate a new **chill** variable (Wind * Temperature)

```
airquality[, chill := Wind * Temp]
```

Calculate the average chill by month and add that as a new variable

```
airquality[, mean.chill := mean(chill) ,
            by = Month]
```

Examples

Change the Wind column from miles per hour to kilometers per hour (1 mph = 1.6 kmh)

```
airquality[, Wind := Wind * 1.6]
```

Calculate a new **chill** variable (Wind * Temperature)

```
airquality[, chill := Wind * Temp]
```

Calculate the average chill by month and add that as a new variable

```
airquality[, mean.chill := mean(chill) ,
            by = Month]
```

Remove the **Ozone** and **Solar.R** columns

```
airquality[, c("Ozone ", "Solar.R ") := NULL]
airquality[, (1:2) := NULL]
```

Merging

Merging

- Combining two datasets is a very routine and important task
- Merging is akin to Joining (in relational database, SQL etc)

Merging

- Combining two datasets is a very routine and important task
- Merging is akin to Joining (in relational database, SQL etc)

- Summary:

$r =$

attr1	attr2
a	r1
b	r2
c	r3

$s =$

attr1	attr3
b	s2
c	s3
d	s4

$r \bowtie s$

attr1	attr2	attr3
b	r2	s2
c	r3	s3

$r \rtimes s$

attr1	attr2	attr3
a	r1	*null*
b	r2	s2
c	r3	s3

$r \ltimes s$

attr1	attr2	attr3
b	r2	s2
c	r3	s3
d	*null*	s4

$r \rtimes\ltimes s$

attr1	attr2	attr3
a	r1	*null*
b	r2	s2
c	r3	s3
d	*null*	s4

Merging Example

Merging Example

- Taken from Stackoverflow webpage
- Create two datasets:
  - `df1 = data.frame(cust_id = c(1:6), Product = c(rep("Toaster", 3), rep("Radio", 3)));`
  - `df2 = data.frame(cust_id = c(2, 4, 6), State = c(rep("Alabama", 2), rep("Ohio", 1)));`
  - `dt1 = as.data.table(df1); dt2 = as.data.table(df2);`

Merging Example

- Taken from Stackoverflow webpage
- Create two datasets:
  - `df1 = data.frame(cust_id = c(1:6), Product = c(rep("Toaster", 3), rep("Radio", 3)));`
  - `df2 = data.frame(cust_id = c(2, 4, 6), State = c(rep("Alabama", 2), rep("Ohio", 1)));`
  - `dt1 = as.data.table(df1); dt2 = as.data.table(df2);`

df1		df2	
cust_id	Product	cust_id	State
1	Toaster	2	Alabama
2	Toaster	4	Alabama
3	Toaster	6	Ohio
4	Radio		
5	Radio		
6	Radio		

Merging Example

- Taken from [Stackoverflow webpage]
- Create two datasets:
  - `df1 = data.frame(cust_id = c(1:6), Product = c(rep("Toaster", 3), rep("Radio", 3)));`
  - `df2 = data.frame(cust_id = c(2, 4, 6), State = c(rep("Alabama", 2), rep("Ohio", 1)));`
  - `dt1 = as.data.table(df1); dt2 = as.data.table(df2);`

- We can merge using `merge()` command on data.table OR we can use a new package `dplyr`
  - The function `merge()` works differently for data.frames and data.tables. It's very slow on DFs and extremely fast on DTs
    - data.table class overrides its own implementation of merge for DTs

df1

cust_id	Product
1	Toaster
2	Toaster
3	Toaster
4	Radio
5	Radio
6	Radio

df2

cust_id	State
2	Alabama
4	Alabama
6	Ohio

Inner Join

- `merge(dt1, dt2, by = "cust_id");`
- `dplyr::inner_join(df1, df2);`

cust_id	Product	State
2	Toaster	Alabama
4	Radio	Alabama
6	Radio	Ohio

Inner Join

- `merge(dt1, dt2, by = "cust_id");`
- `dplyr::inner_join(df1, df2);`

cust_id	Product	State
2	Toaster	Alabama
4	Radio	Alabama
6	Radio	Ohio

Full Join

- `merge(dt1, dt2, by = "cust_id", all = T);`
- `dplyr::full_join(df1, df2);`

cust_id	Product	State
1	Toaster	NA
2	Toaster	Alabama
3	Toaster	NA
4	Radio	Alabama
5	Radio	NA
6	Radio	Ohio

Left Join

- `merge(dt1, dt2, by = "cust_id", all.x = T);`
- `dplyr::left_join(df1, df2);`

cust_id	Product	State
1	Toaster	NA
2	Toaster	Alabama
3	Toaster	NA
4	Radio	Alabama
5	Radio	NA
6	Radio	Ohio

Left Join

- `merge(dt1, dt2, by = "cust_id", all.x = T);`
- `dplyr::left_join(df1, df2);`

cust_id	Product	State
1	Toaster	NA
2	Toaster	Alabama
3	Toaster	NA
4	Radio	Alabama
5	Radio	NA
6	Radio	Ohio

Right Join

- `merge(dt1, dt2, by = "cust_id", all.y = T);`
- `dplyr::right_join(df1, df2);`

cust_id	Product	State
2	Toaster	Alabama
4	Radio	Alabama
6	Radio	Ohio

Cartesian Product

- Every row of `df1` multiplied with every row of `df2`

Cartesian Product

- Every row of `df1` multiplied with every row of `df2`
- `cart_prod = dt1[, as.list(dt2), by = "cust_id"];`
- `cart_prd = merge(df1, df2, by = NULL);`

S.No.	cust_id.x	Product	cust_id.y	State
1	1	Toaster	2	Alabama
2	2	Toaster	2	Alabama
3	3	Toaster	2	Alabama
4	4	Radio	2	Alabama
5	5	Radio	2	Alabama
6	6	Radio	2	Alabama
7	1	Toaster	4	Alabama
8	2	Toaster	4	Alabama
9	3	Toaster	4	Alabama

S.No.	cust_id.x	Product	cust_id.y	State
10	4	Radio	4	Alabama
11	5	Radio	4	Alabama
12	6	Radio	4	Alabama
13	1	Toaster	6	Ohio
14	2	Toaster	6	Ohio
15	3	Toaster	6	Ohio
16	4	Radio	6	Ohio
17	5	Radio	6	Ohio
18	6	Radio	6	Ohio

- Cartesian products are extremely slow. Never do that even on a decent sized (> 1e4 rows) dataset. Your computer will probably hang.
- Although there is no use of Cartesian products, it encapsulates all types of merges. Meaning we can extract any type of merge from a Cartesian product.
- Inner join can be extracted via
  - `idx = which(cart_prd$cust_id.x == cart_prd$cust_id.y);`
  - `cart_prd[idx,];`