Dataframe manipulations

Reading, processing and analysing dataframes

Michiel Noback

19 November 2015

`apply()` and its relatives

Intro

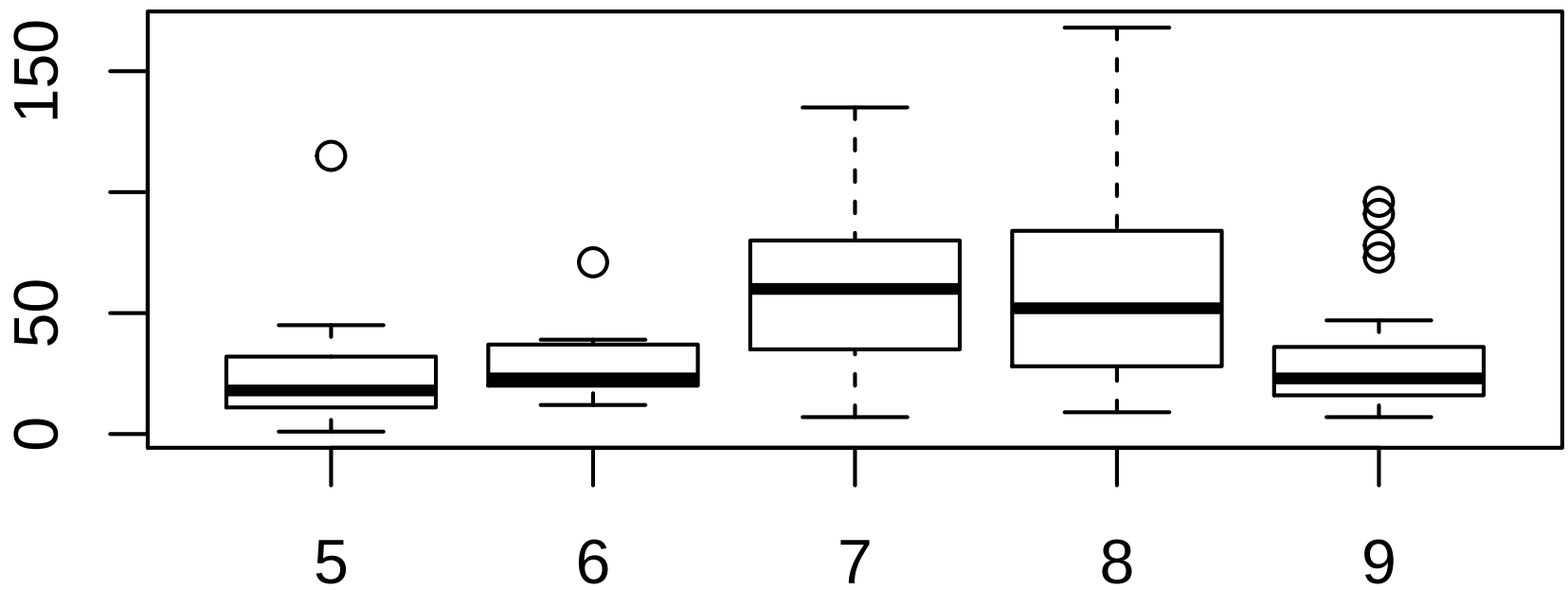
Dataframes are ubiquitous in data analyses using R. There are many functions tailored for DF manipulations; you have already seen `cbind()` and `rbind()`. In this presentation, we'll explore a few new functions and techniques for working with DFs.

- `with()`
- `subset()`
- `apply()` and its relatives
- `aggregate()`
- `split()`

with()

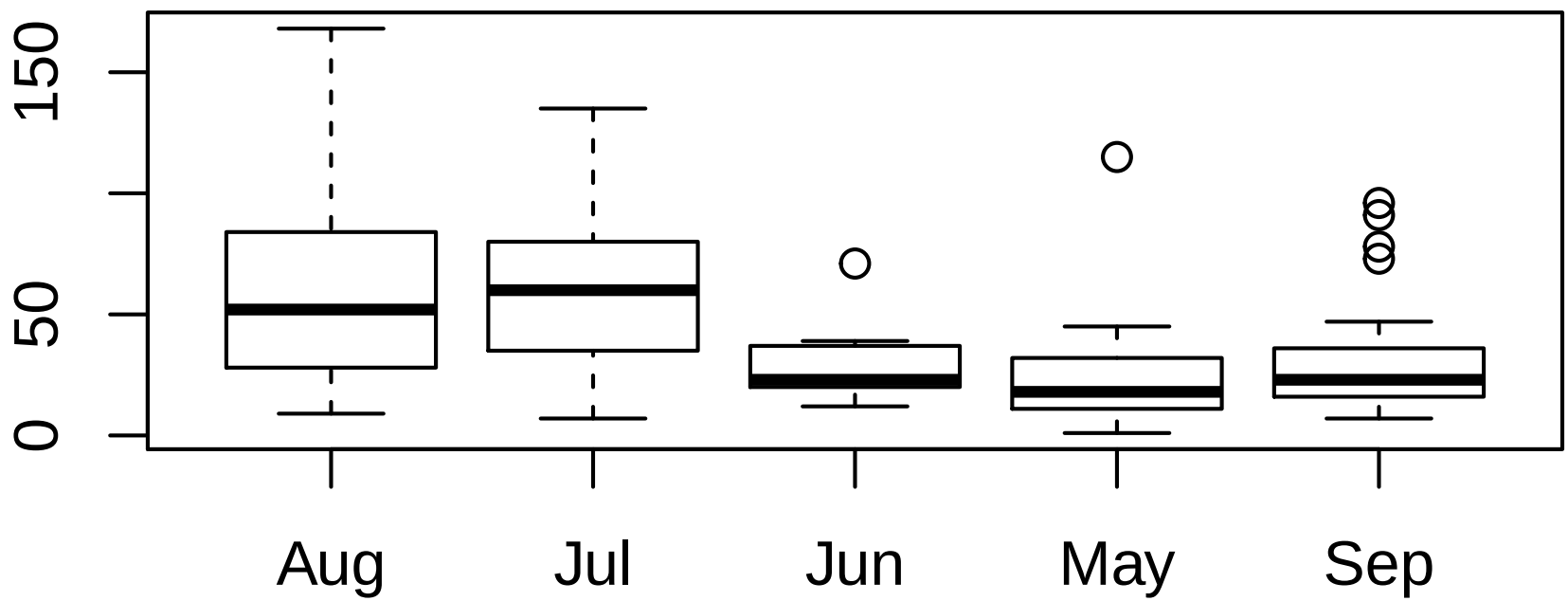
- Evaluate an expression in a data environment

```
## this reads really well
with(airquality, boxplot(Ozone ~ Month))
## as opposed to
#boxplot(airquality$Ozone ~ airquality$Month)
```



- You can also combine expressions within a block of curly braces.

```
with(
    airquality, {
        month <- factor(month.abb[Month])
        boxplot(Ozone ~ month)})
```



subset()

Return subsets of vectors, matrices or data frames which meet conditions.

```
head(
    ## same as airquality[ airquality$Temp >= 80, ]
    subset(airquality, Temp >= 80),
    n = 4
)
```

```
##    Ozone Solar.R Wind Temp Month Day
## 29    45     252 14.9   81     5  29
## 35    NA     186  9.2   84     6   4
## 36    NA     220  8.6   85     6   5
## 38    29     127  9.7   82     6   7
```

```
head(
    subset(airquality, Temp >= 80, select = c(Ozone, Temp))
    n = 4
)
```

```
##    Ozone Temp
## 29    45   81
## 35    NA   84
## 36    NA   85
## 38    29   82
```

```r
head(
    subset(airquality, is.na(Ozone), select = c(Ozone : Wi
    n = 4
)
```

```
##    Ozone Solar.R Wind
## 5     NA      NA 14.3
## 10    NA     194  8.6
## 25    NA      66 16.6
## 26    NA     266 14.9
```

`apply()` and its relatives

The apply family

- When you want to do something with
  - all rows or all columns of a dataframe
  - all values in a vector
  - all elements in a list
- Looping with `for` is very tempting, but often highly inefficient

Overview

- `apply`: Apply a function over the margins of an array
- `lapply`: Loop over a list and evaluate a function on each element; returns a list of the same length
- `sapply`: Same as lapply but try to simplify the result
- `tapply`: Apply a function over subsets of a vector
- There are more but these are the important ones

apply()

- apply *mean* to all columns of the built-in `cars` dataset
- apply needs to know
  1. what DF to apply to
  2. over which margin (columns or rows)
  3. what function to apply

```
apply(cars, 2, mean) # apply over columns
```

```
## speed  dist
## 15.40 42.98
```

apply(): calculate the BMI

- BMI is calculated as $(weight/height^2) * 703$ where weight is in pounds and height in inches.

```
head(women, n=3)
```

```
##   height weight
## 1     58    115
## 2     59    117
## 3     60    120
```

```
women$bmi <- apply(women, 1, function(x) (x[2] / x[1]^2) *
head(women, n=3)
```

```
##   height weight      bmi
## 1     58    115 24.03240
## 2     59    117 23.62856
## 3     60    120 23.43333
```

- It is not considered good practice to use inline (anonymous) functions because
  - Inline functions make your code less readable
  - Inline functions can not be re-used
- Here is the previous example refactored

```
bmi <- function(heightWeight) {
    (heightWeight[2] / heightWeight[1]^2) * 703
}
women$bmi <- apply(women, 1, bmi)
head(women, n=4)
```

```
##   height weight      bmi
## 1     58    115 24.03240
## 2     59    117 23.62856
## 3     60    120 23.43333
## 4     61    123 23.23811
```

- or, maybe better because more generic, but without apply.

```
bmi <- function(height, weight) {
    (weight / height^2) * 703
}
women$bmi2 <- bmi(women$height, women$weight)
head(women, n=4)
```

```
##   height weight      bmi     bmi2
## 1     58    115 24.03240 24.03240
## 2     59    117 23.62856 23.62856
## 3     60    120 23.43333 23.43333
## 4     61    123 23.23811 23.23811
```

passing arguments to the applied function

- Sometimes, the applied function needs to have other arguments passed
- The ... argument to apply makes this possible (type ?apply)

```
#sum and power up
spwr <- function(x, p = 2) {sum(x)^p}
df <- data.frame(a = 1:5, b = 6:10)
df
```

```
##   a  b
## 1 1  6
## 2 2  7
## 3 3  8
## 4 4  9
## 5 5 10
```

```r
apply(df, 1, spwr) # spwr will use default value for p (p =
apply(df, 1, spwr, p = 3) #pass power p = 3 to function spu
```

```
## [1]  49  81 121 169 225
## [1]  343  729 1331 2197 3375
```

lapply(): apply to a list

- lapply() applies a function to all elements of a list and returns a list with the same length, each element the result of applying the function

```
myNumbers = list(
    one = c(1, 3, 4),
    two = c(3, 2, 6, 1),
    three = c(5, 7, 6, 8, 9))
lapply(myNumbers, mean)
```

```
## $one
## [1] 2.666667
##
## $two
## [1] 3
##
## $three
## [1] 7
```

- Same, but with `sqrt()` applied
- Note how the nature of the applied function influences the way it is treated

```
lapply(myNumbers, sqrt)
```

```
## $one
## [1] 1.000000 1.732051 2.000000
##
## $two
## [1] 1.732051 1.414214 2.449490 1.000000
##
## $three
## [1] 2.236068 2.645751 2.449490 2.828427 3.000000
```

sapply(): apply to a list and try to simplify

- When using the same example as above, but with sapply, you get a vector returned
- Note that the elements of the vector do have names attached

```
myNumbers = list(
    one = c(1, 3, 4),
    two = c(3, 2, 6, 1),
    three = c(5, 7, 6, 8, 9))
sapply(myNumbers, mean)
```

```
##       one       two     three
## 2.666667 3.000000 7.000000
```

tapply(): split and apply

- tapply(): Apply a function over subsets of a vector
- in human language: split a vector into groups according to a the levels in a second vector and apply the given function to each group

```
tapply(chickwts$weight, chickwts$feed, mean)
```

```
##    casein horsebean   linseed  meatmeal   soybean sunflo
##  323.5833  160.2000  218.7500  276.9091  246.4286  328.9
```

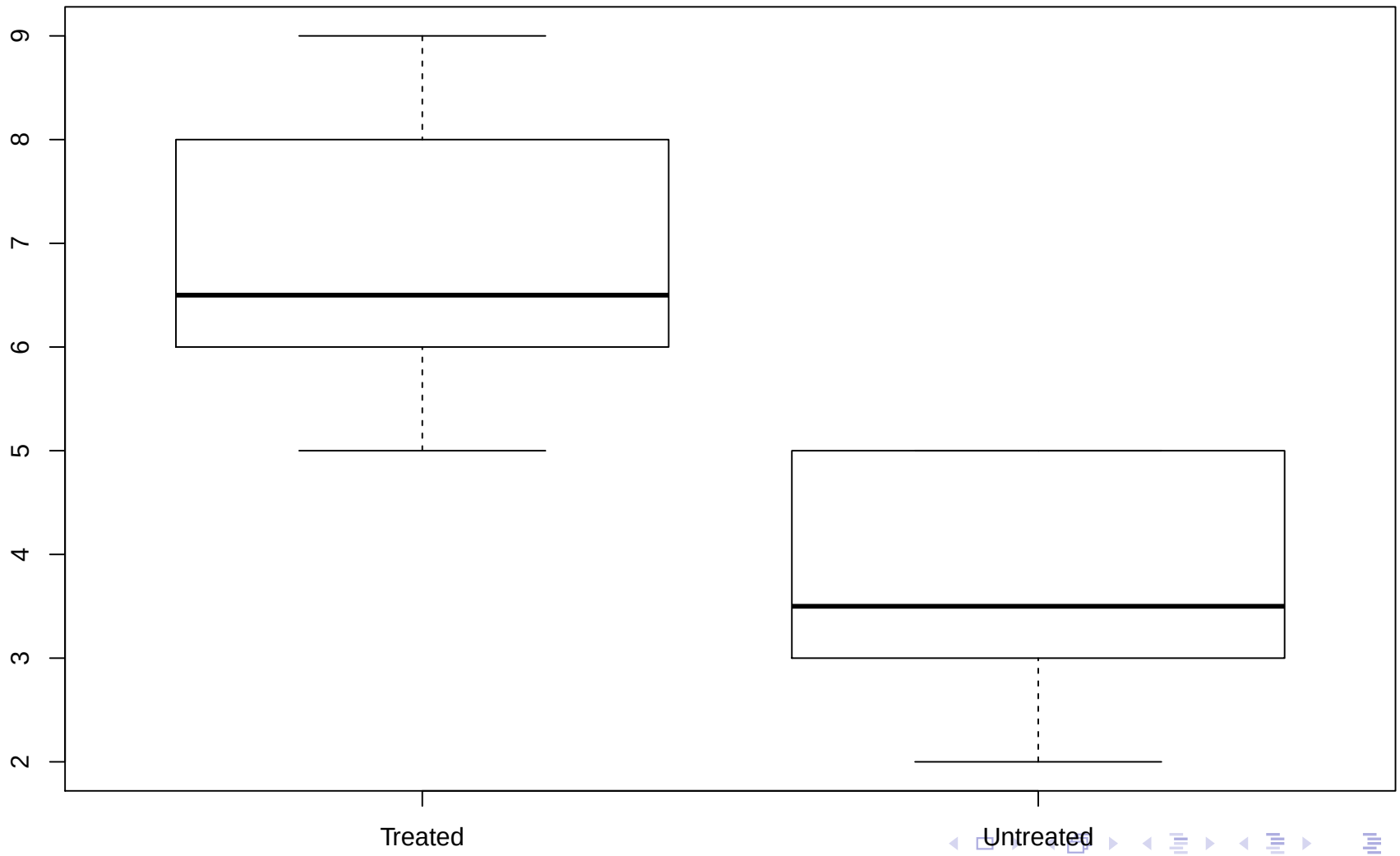
split(): split into groups

- Use split() when a dataframe needs to be divided depending on the value of some grouping variable. The result is a list, with a member for each grouping value
- Here we have the response of Treated (T) and Untreated (UT) subjects

```
myData <- data.frame(
    response = c(5, 8, 4, 5, 9, 3, 6, 7, 3, 6, 5, 2),
    treatment = factor(
        c("UT", "T", "UT", "UT", "T", "UT", "T", "T", "UT",
splData <- split(myData, myData$treatment)
str(splData)
```

```
## List of 2
##  $ T :'data.frame':  6 obs. of  2 variables:
##   ..$ response : num [1:6] 8 9 6 7 6 5
##   ..$ treatment: Factor w/ 2 levels "T","UT": 1 1 1 1 1
##  $ UT:'data.frame':  6 obs. of  2 variables:
##   ..$ response : num [1:6] 5 4 5 3 3 2
```

```
## this trivial example could also have been done with
## boxplot(myData$response ~ myData$treatment)
boxplot(splData$T$response, splData$UT$response,
        names = c("Treated", "Untreated"))
```



aggregate(): Compute summary statistics of subsets

Splits the data into subsets, computes summary statistics for each, and returns the result in a convenient form.

```
aggregate(Temp ~ Month, data = airquality, mean)
```

```
##   Month     Temp
## 1     5 65.54839
## 2     6 79.10000
## 3     7 83.90323
## 4     8 83.96774
## 5     9 76.90000
```

many roads lead to Rome

The statements on the next three slides are all essentially the same

```
aggregate(chickwts$weight, by = list(chickwts$feed), FUN =
```

```
##     Group.1        x
## 1    casein 323.5833
## 2 horsebean 160.2000
## 3   linseed 218.7500
## 4  meatmeal 276.9091
## 5   soybean 246.4286
## 6 sunflower 328.9167
```

```
aggregate(weight ~ feed, data = chickwts, FUN = mean)
```

```
##         feed   weight
## 1    casein 323.5833
## 2 horsebean 160.2000
## 3   linseed 218.7500
## 4  meatmeal 276.9091
```

```
tapply(chickwts$weight, chickwts$feed, mean)
```

```
##    casein horsebean   linseed  meatmeal   soybean sunflo
##  323.5833  160.2000  218.7500  276.9091  246.4286  328.9
```

```
with(chickwts, tapply(weight, feed, mean))
```

```
##    casein horsebean   linseed  meatmeal   soybean sunflo
##  323.5833  160.2000  218.7500  276.9091  246.4286  328.9
```

```
sapply(split(chickwts, chickwts$feed), function(x){mean(x$w
```

```
##    casein horsebean   linseed  meatmeal   soybean sunflo
##  323.5833  160.2000  218.7500  276.9091  246.4286  328.9
```

```
library(dplyr)
group_by(chickwts, feed) %>% summarise(m = mean(weight))
```

```
## Source: local data frame [6 x 2]
##
##         feed        m
##       (fctr)    (dbl)
## 1     casein 323.5833
## 2 horsebean 160.2000
## 3    linseed 218.7500
## 4   meatmeal 276.9091
## 5    soybean 246.4286
## 6 sunflower 328.9167
```

`merge()` to bring data from two dataframes together

- In many cases, data is distributed over multiple sources (often files).
- The `merge()` function helps you combining these datasets on common identifiers.
- suppose you want to analyse gene class in relation to expression levels
- Note that if two columns have the same name, merge() uses these by default!

```r
gene.classes <- data.frame(
    geneID = c("gi:267", "gi:235", "gi:332"),
    class = c("regulator", "metabolism", "structural"))
top.expressed.genes <- data.frame(
    tissue = c("connective", "muscle", "nervous", "epitheli
    gene = c("gi:235", "gi:267", "gi:235", "gi:332"))
merge(top.expressed.genes, gene.classes, by.x="gene", by.y=
```

```
##      gene     tissue      class
## 1 gi:235 connective metabolism
## 2 gi:235    nervous metabolism
## 3 gi:267     muscle  regulator
## 4 gi:332 epithelial structural
```