Manipulating Data

Programming in R for Data Science

Anders Stockmarr, Kasper Kristensen, Anders Nielsen

Manipulating data vs. working with data

- When working with data, we leave them unchanged.
- Manipulating data changes data.

Example: Data transformation.

```
log.airquality<-log(airquality)
summary(log.airquality)
```

```
     Ozone           Solar.R           Wind             Temp           Month            Day
 Min.   :0.000   Min.   :1.946   Min.   :0.5306   Min.   :4.025   Min.   :1.609   Min.   :0.000
 1st Qu.:2.890   1st Qu.:4.751   1st Qu.:2.0015   1st Qu.:4.277   1st Qu.:1.792   1st Qu.:2.079
 Median :3.450   Median :5.323   Median :2.2721   Median :4.369   Median :1.946   Median :2.773
 Mean   :3.419   Mean   :5.008   Mean   :2.2272   Mean   :4.347   Mean   :1.924   Mean   :2.507
 3rd Qu.:4.147   3rd Qu.:5.556   3rd Qu.:2.4423   3rd Qu.:4.443   3rd Qu.:2.079   3rd Qu.:3.135
 Max.   :5.124   Max.   :5.811   Max.   :3.0301   Max.   :4.575   Max.   :2.197   Max.   :3.434
 NA's   :37      NA's   :7
```

Summary statistics: sapply() and lapply()

Summary statistics: Create your own.

```
> my.summary<-function(x){data.frame(Min=min(x,na.rm=TRUE),
+                                    Median=median(x,na.rm=TRUE),
+                                    Mean=mean(x,na.rm=TRUE),
+                                    Max=max(x,na.rm=TRUE))}
> sapply(airquality,my.summary)
```

	Ozone	Solar.R	Wind	Temp	Month	Day
Min	1	7	1.7	56	5	1
Median	31.5	205	9.7	79	7	16
Mean	42.12931	185.9315	9.957516	77.88235	6.993464	15.80392
Max	168	334	20.7	97	9	31

tapply(), aggregate() and by(): Apply a function within a group

- Consider the following data frame:

```
> dat
   gender height
1    Male     10
2    Male      5
3    Male     12
4    Male     10
5    Male      2
6  Female      7
7  Female      6
8  Female     12
9  Female      9
10 Female      4
```

Calculating means by group

- Three ways to calculate group means:

```
> tapply(dat$height, dat$gender, mean)
  Male Female
   7.8    7.6

> aggregate(height~gender, data=dat, mean)
  gender height
1   Male    7.8
2 Female    7.6

> by(dat$height, dat$gender, mean)
dat$gender: Male
[1] 7.8
---------------------------------------------------------------------
dat$gender: Female
[1] 7.6
```

Grouped means with several classification variables

- Now consider the expanded data:

```
> dat2
   gender     tmt height
1    Male  active     10
2    Male placebo      5
3    Male  active     12
4    Male placebo     10
5    Male  active      2
6  Female placebo      7
7  Female  active      6
8  Female placebo     12
9  Female  active      9
10 Female placebo      4
```

- classifiers: `Gender` and `treatment`.

Grouped means with several classification variables

tapply(dat2$height, list(dat2$gender,dat2$tmt), mean)

```
       active  placebo
Male      8.0 7.500000
Female    7.5 7.666667
```

aggregate(height~gender+tmt, data=dat2, mean)

```
  gender     tmt   height
1   Male  active 8.000000
2 Female  active 7.500000
3   Male placebo 7.500000
4 Female placebo 7.666667
```

by(dat2$height, list(dat2$gender, dat2$tmt), mean)

```
: Male
: active
[1] 8
----------------------------------------------------------------------------
: Female
: active
[1] 7.5
----------------------------------------------------------------------------
: Male
: placebo
[1] 7.5
----------------------------------------------------------------------------
: Female
: placebo
[1] 7.666667
```

Run times for group means

The different way that R performs the calculations has an impact for large datasets.

```
> x<-rnorm(1000000)
> y<-sample(1:1000,1000000,replace=T)
> groups<-paste("group",1:1000)
> my.data<-data.frame(data=rnorm(1000000),
+                     group=sample(groups,1000000,replace=T))
```

`my.data` contains one million data points, grouped into 1000 groups.

Run times for group means

```
> # tapply():
> time0<-Sys.time()
> tapply(my.data$data,my.data$group,mean)
> time1<-as.numeric(Sys.time()-time0)

> # aggregate():
> time0<-Sys.time()
> aggregate(data~group,mean,data=my.data)
> time2<-as.numeric(Sys.time()-time0)

> # by():
> time0<-Sys.time()
> by(my.data$data,my.data$group,mean)
> time3<-as.numeric(Sys.time()-time0)

>  my.runtime<-data.frame(tapply=c(time1,time1/time1),
+                          aggregate=c(time2,time2/time1),
+                          by=c(time3,time3/time1))
>  rownames(my.runtime)<-c("Time elapsed:","Relative to tapply():")

> my.runtime

                      tapply aggregate        by
Time elapsed:         0.1248  1.396007 0.2349999
Relative to tapply(): 1.0000 11.185957 1.8830125
```

Attaching and detaching data

Attach a data frame to R's search path with the `attach()` function:

```
> tapply(dat$height, dat$gender, mean)
  Male Female
   7.8    7.6
> attach(dat)
> tapply(height, gender, mean)
  Male Female
   7.8    7.6
```

Remember to `detach()` when you are done:

```
> detach(dat)
```

Masking R objects

Similar identifiers in the R memory OVERRIDES adding an R object to the R search path with attach():

```
> x1<-1:3
> my.data<-data.frame(x1=4:6,x2=7:9)
> attach(my.data)
```

You can't refer to x1 in my.data:

```
> cbind(x1,x2)

     x1 x2
[1,]  1  7
[2,]  2  8
[3,]  3  9
```

Adding a new object OVERRIDES the previous addition to the search path:

```
> my.data2<-data.frame(x1=10:12,x2=13:15)
> attach(my.data2)

> cbind(x1,x2)

     x1 x2
[1,]  1 13
[2,]  2 14
[3,]  3 15
```

R's search path

An attached object is placed on top of R's seach path, but below the Global R memory!
You can see the hierachy of R searches for identifiers at any time, with the searchpaths() function:

The top of R's search path():

```
> searchpaths()[1:3]
```

```
[1] ".GlobalEnv" "my.data2"    "my.data"
```

R takes `x1` from the Global Environment (the R memory), and `x2` from `my.data2`.

Let us clean up:

```
> detach(my.data,my.data2)
```

The with() function

Lets recreate the messy situation:

```
> x1<-1:3
> my.data<-data.frame(x1=4:6,x2=7:9)
> my.data2<-data.frame(x1=10:12,x2=13:15)
> attach(my.data)
> attach(my.data2)
```

with() temporarily puts the first argument in the top of R's search hierachy:

```
> sum.and.dif<-with(my.data,cbind(x1+x2,x1-x2))
> sum.and.dif

     [,1] [,2]
[1,]   11   -3
[2,]   13   -3
[3,]   15   -3

> cbind(x1+x2,x1-x2)

     [,1] [,2]
[1,]   14  -12
[2,]   16  -12
[3,]   18  -12
```

Let us clean up again:

```
> detach(my.data,my.data2)
```

Tabulating data: table() and xtabs()

Functions used for tabulating cross-referenced data. The most important functions to apply are (incl. table() and xtabs()):

Function	Description
table(var1, var2, ...,varN)	Creates an N-way contingency table of counts from categorical variables
xtabs(formula, data)	Creates an N-way contingency table based on a formula and data
prop.table(table,*margins*)	Expresses entries in the table as fractions of the marginal table defined by *margins*
margin.table(table,*margins*)	Computes the sum of table entries for a marginal table defined by *margins*
addmargins(table,*margins*)	Adds sums to the margins of a table
ftable(table)	creates a flat contingency table

Example: table() and the airquality data

We investigate the number of days in a month with high levels of ozone:

```
> my.table<-with(airquality,table(OzHi = Ozone > 80, Month))
> my.table

       Month
OzHi     5  6  7  8  9
  FALSE 25  9 20 19 27
  TRUE   1  0  6  7  2

> my.table.2<-addmargins(my.table,1:2)
> my.table.2

       Month
OzHi      5   6   7   8   9 Sum
  FALSE  25   9  20  19  27 100
  TRUE    1   0   6   7   2  16
  Sum    26   9  26  26  29 116
```

Example: table() and the airquality data

Converting to frequencies with `prop.table()`:

```
> my.table.3<-prop.table(my.table,2)
> my.table.3<-addmargins(my.table.3,1)
> my.table.3

       Month
OzHi             5          6          7          8          9
  FALSE 0.96153846 1.00000000 0.76923077 0.73076923 0.93103448
  TRUE  0.03846154 0.00000000 0.23076923 0.26923077 0.06896552
  Sum   1.00000000 1.00000000 1.00000000 1.00000000 1.00000000
```

Converting to percentages: Multiplying with 100 and rounding:

```
> round(100*my.table.3)

       Month
OzHi      5   6   7   8   9
  FALSE  96 100  77  73  93
  TRUE    4   0  23  27   7
  Sum   100 100 100 100 100
```

Example: xtabs() and Admissions to Berkeley

Data from UC Berkeley admissions, 1973:

```
> DF <- as.data.frame(UCBAdmissions)
> head(DF)
     Admit Gender Dept Freq
1 Admitted   Male    A  512
2 Rejected   Male    A  313
3 Admitted Female    A   89
4 Rejected Female    A   19
5 Admitted   Male    B  353
6 Rejected   Male    B  207
```

DF contains information on Admission(admitted/not admitted), Gender, Department and Frequency.

Example: xtabs() and Admissions to Berkeley

Organizing data as a contingency table with xtabs:

```
> DF <- as.data.frame(UCBAdmissions)
> head(DF)
```

```
     Admit Gender Dept Freq
1 Admitted   Male    A  512
2 Rejected   Male    A  313
3 Admitted Female    A   89
4 Rejected Female    A   19
5 Admitted   Male    B  353
6 Rejected   Male    B  207
```

```
> mytable <- xtabs(Freq ~  Gender + Admit + Dept, data=DF)
> ftable(mytable)
```

```
              Dept   A   B   C   D   E   F
Gender Admit
Male   Admitted    512 353 120 138  53  22
       Rejected    313 207 205 279 138 351
Female Admitted     89  17 202 131  94  24
       Rejected     19   8 391 244 299 317
```

Marginal table: Gender vs. admission

We tabulate gender vs. admission with `margin.table()`:

```
> margin.table(mytable,1:2)
        Admit
Gender   Admitted Rejected
  Male       1198     1493
  Female      557     1278
```

Converting it to frequencies:

```
> prop.table(margin.table(mytable,1:2),1)
        Admit
Gender    Admitted  Rejected
  Male   0.4451877 0.5548123
  Female 0.3035422 0.6964578
```

Males appear to do better than Females.

Marginal table: Admission vs. department

We tabulate admission vs. department:

```
> prop.table(margin.table(mytable,2:3),1)
          Dept
Admit               A          B          C          D          E
  Admitted 0.34245014 0.21082621 0.18347578 0.15327635 0.08376068 0.026
  Rejected 0.11981234 0.07758932 0.21508481 0.18874053 0.15770480 0.241
```

Apparently, Department E and F are the hardest to be admitted into, while Department A and B are the easiest.

Marginal table: Gender vs. department

We tabulate gender vs. department:

```
> prop.table(margin.table(mytable,c(1,3)),1)
        Dept
Gender            A          B          C          D          E
  Male   0.30657748 0.20810108 0.12077295 0.15496098 0.07097733 0.13861
  Female 0.05885559 0.01362398 0.32316076 0.20435967 0.21416894 0.18583
```

Males tend to apply to Department A and B, while Females tend to apply to department C-D. Department A and B are the easiest to get into...

Marginal table: Investigation by department

Data for department A can be extracted as

```
> DepA<-mytable[,,1]
> ftable(DepA)
```

```
       Admit Admitted Rejected
Gender
Male               512      313
Female              89       19
```

```
> prop.table(DepA,1)
```

```
        Admit
Gender    Admitted  Rejected
  Male   0.6206061 0.3793939
  Female 0.8240741 0.1759259
```

Females have higher admission rate for Department A.
In fact, Females have higher admission rates in 4 out of 6 departments.

The dplyr package

The *dplyr* package by Hadley Wickham contains a framework for data manipulation in R.

Used in a number of Microsoft courses.