

Divide et impera: Split-apply-combine-**Ansätze** in R

Bernd Weiß

<http://berndweiss.net>

7. Treffen der Köln R User Group
am 18. Oktober 2013



Gliederung

Problem

Übersicht von Paketen und Befehlen

Statistische Funktionen (Aggregation)

Base R

Das `plyr`-Paket

Datenaufbereitung

No loops, please!

Weitere Quellen und Literatur

Gliederung

Problem

Übersicht von Paketen und Befehlen

Statistische Funktionen (Aggregation)

Base R

Das `plyr`-Paket

Datenaufbereitung

No loops, please!

Weitere Quellen und Literatur

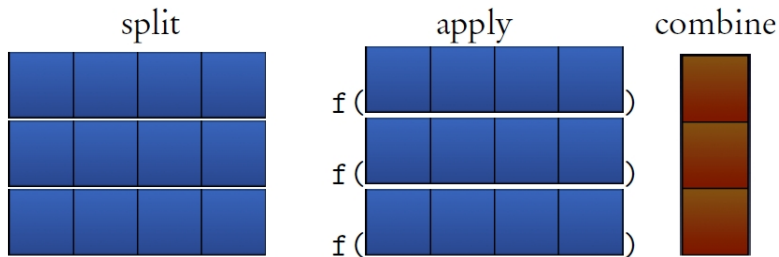
Beispielprobleme

- Berechnen des mittleren Einkommens nach dem Gruppierungsmerkmal höchster Schulabschluss.
- Schätzen eines linearen Regressionsmodells für verschiedene Altersgruppen.
- Erstellen einer Tabelle mit (beliebigen) deskriptiven Statistiken für einen Datensatz (gesamt oder wieder für verschiedene Gruppen).
- Simulation von 1000 Datensätzen, schätze für jeden Datensatz ein lineares Regressionsmodell, extrahiere die Regressionskoeffizienten und untersuche die Verteilung der Koeffizienten.
- In einem Verzeichnis befinden sich 2000 csv-Dateien. Diese sollen eingelesen werden, berechne pro Datei einen Mittelwert und speichere den neuen Datensatz mit 2000 Mittelwerten ab.
- ...

Was heißt „split-apply-combine“?

- „split“: Teile ein großes Problem in kleinere Teile (manchmal liegen die Einzelteile aber auch schon vor).
- „apply“: Wende eine beliebige Funktion auf jedes Teil an.
- „combine“: Fasse die Ergebnisse der einzelnen Funktionen wieder zusammen.

Das „split-apply-combine“-Prinzip



(Quelle: Shalizi, [2013](#))

Das Problem und die Lösung in R: „vectorization“

- „split-apply-combine“-Probleme werden in anderen Sprachen üblicherweise mit Schleifen gelöst.
- Schleifen in R vermeiden, stattdessen „vectorization“¹ (entsprechende Funktionen sind in C implementiert und damit häufig schneller als purer R-Code). Teetor (2011, S. 147) schreibt dazu:

„Where traditional programming languages use loops, R uses vectorized operations and the apply functions to crunch data in batches, greatly streamlining the calculations.“

- „Vectorized“ R-Code ist (häufig) lesbarer.
- „Vectorized functions“ können parallelisiert werden.

¹ „Vectorization: Where functions are applied element-wise to vectors.“ (Matloff, 2011, S. 25)

Gliederung

Problem

Übersicht von Paketen und Befehlen

Statistische Funktionen (Aggregation)

Base R

Das `plyr`-Paket

Datenaufbereitung

No loops, please!

Weitere Quellen und Literatur

Gliederung der Pakete und Befehle

Alle nachfolgend aufgeführten Pakete und Befehle können nach folgenden Dimensionen/Aspekten gegliedert werden:

- Statistische Aggregation ($Dim_{input} > Dim_{output}$) vs Datentaufbereitung/-transformation ($Dim_{input} = Dim_{output}$)
- Ausgangsdatentyp (z.B. `data.frame`) vs Zieldatentyp (z.B. `vector`)
- (Vektorisierte) Alternative zu Schleifen vs Aggregation/Transformation

Eine (unvollständige) Übersicht von Paketen und Befehlen

- `base`-Paket:
 - `apply`, `lapply` (`sapply`), `tapply`, `mapply`, ...
 - `by`, `aggregate`
 - `replicate`
 - ...
- `plyr`-Paket (u.a. konsistentes Namensschema für Ein- und Ausgabedatentyp, parallele Verarbeitung)
- `doBy`-Paket (orientiert sich an SAS-Funktion, stat. Aggregation)
- `data.table`-Paket (meistens schnellste² Lösung, aber eigenwillige Syntax)
- ...

Für eine umfassendere Darstellung siehe u.a., Wickham (2011).

²Für einen Geschwindigkeitstest siehe [A speed test comparison of plyr, data.table, and dplyr](#), abgerufen am 2.10.2013

Base R Funktionen

- Viele dieser „split-apply-combine“-Funktionen im `base`-Paket unterscheiden sich lediglich nach dem Eingabe- und Ausgabebetyp.
- Eine (unvollständige) Übersicht gibt die folgende Tabelle:

Function	Input	Output	Anmerkung
<code>apply</code>	array, matrix	array, list	Funktion bezieht sich auf Dimensionen („margins“) eines array
<code>lapply</code>	list, data.frame	list	
<code>aggregate</code>	ts, data.frame	ts, data.frame	Formelnotation möglich
<code>tapply</code>	vector	array	column names werden gelöscht
<code>by</code>	data.frame, matrix	list, array	wrapper für <code>tapply</code>

Quelle: <http://stackoverflow.com/a/3506108/403473>; eigene Darstellung

Gliederung

Problem

Übersicht von Paketen und Befehlen

Statistische Funktionen (Aggregation)

Base R

Das `plyr`-Paket

Datenaufbereitung

No loops, please!

Weitere Quellen und Literatur

Abschnittsübersicht

Problem

Übersicht von Paketen und Befehlen

Statistische Funktionen (Aggregation)

Base R

Das `plyr`-Paket

Datenaufbereitung

No loops, please!

Weitere Quellen und Literatur

apply() I

Problem

Eine Funktion (mean, sd, etc.) auf alle Spalten oder Zeilen eines Datensatzes anwenden.

```
## Datensatz erstellen
dat <- data.frame(v1 = sample(1:10, 5,
                             replace = TRUE),
                  v2 = sample(0:1, 5,
                             replace = TRUE))

## Einen missing value einbauen
dat$v1[2] <- NA
dat
```

	v1	v2
1	5	1
2	NA	1
3	10	0
4	10	0
5	4	1

apply() II

```
## Fuer 2d-array:  
## MARGIN = 1: rowwise, MARGIN = 2: columnwise  
##  
## Spaltenmittelwerte  
apply(dat, MARGIN = 2, mean, na.rm = TRUE)
```

```
      v1      v2  
7.25 0.60
```

```
## Zeilenmittelwerte  
apply(dat, MARGIN = 1, mean, na.rm = TRUE)
```

```
[1] 3.0 1.0 5.0 5.0 2.5
```

```
## Anzahl missing values  
apply(dat, 2, FUN = function(x)sum(is.na(x)))
```

```
v1 v2  
1  0
```

aggregate() I

Problem

Eine Funktion (mean, sd, etc.) auf Subgruppen eines Datensatzes anwenden und Aggregatstatistiken berechnen.

```
## Datensatz erstellen
dat <- data.frame(g1 = c(1, 1, 1, 2, 2, 2),
                  g2 = c(1, 1, 2, 2, 3, 3),
                  ## M_1 = 2, M_2 = 1
                  x = c(1, 2, 3, 0.5, 1, 1.5))
```

dat

	g1	g2	x
1	1	1	1.0
2	1	1	2.0
3	1	2	3.0
4	2	2	0.5
5	2	3	1.0
6	2	3	1.5

aggregate() II

```
## Gruppenmittelwerte berechnen
```

```
aggregate(dat$x, by = list(dat$g1, dat$g2), mean)
```

	Group.1	Group.2	x
1	1	1	1.50
2	1	2	3.00
3	2	2	0.50
4	2	3	1.25

```
aggregate(. ~ g1 + g2, dat, mean)
```

	g1	g2	x
1	1	1	1.50
2	1	2	3.00
3	2	2	0.50
4	2	3	1.25

tapply() I

Problem

Eine Funktion (mean, sd, etc.) auf Subgruppen eines Datensatzes anwenden und Aggregatstatistiken berechnen.

```
## Datensatz erstellen
dat <- data.frame(
  g1 = factor(c("A", "A", "A", "B", "B", "B")),
  g2 = c(1, 1, 2, 2, 3, 3),
  ## M_1 = 2, M_2 = 1
  x = c(1, 2, 3, 0.5, 1, 1.5))
```

dat

	g1	g2	x
1	A	1	1.0
2	A	1	2.0
3	A	2	3.0
4	B	2	0.5
5	B	3	1.0
6	B	3	1.5

tapply() II

```
## Gruppenmittelwerte berechnen  
## (missings entsprechen fehlenden Gruppen)  
tapply(dat$x, INDEX = list(dat$g1, dat$g2), mean)
```

	1	2	3
A	1.5	3.0	NA
B	NA	0.5	1.25

Abschnittsübersicht

Problem

Übersicht von Paketen und Befehlen

Statistische Funktionen (Aggregation)

Base R

Das `plyr`-Paket

Datenaufbereitung

No loops, please!

Weitere Quellen und Literatur

Warum das `plyr`-Paket nutzen?

- Konsistentes Namensschema (`{Input}{Output}ply` , bspw. `ddply`)
- Teilweise schneller als `base`-Funktionen
- Parallelisierung möglich
- Speichereffizienter als `base`-Funktionen

(siehe ausführlich Wickham, 2011).

Hinweise zur Nutzung

- Der exakte Funktionsaufruf hängt vom Typ des Eingabeobjektes (Input) ab:

- `a*ply(.data, .margins, .fun, ...)`

- `d*ply(.data, .variables, .fun, ...)`

- `l*ply(.data, .fun, ..., ...)`

- (Ist die Ausgabe ein data frame, sollte man angeben, ob man Daten aggregieren (via `summarize`) oder transformieren (via `transform`) möchte.)

(siehe ausführlich Wickham, 2011).

plyr: Deskriptive Statistiken I

Problem

Eine Tabelle mit (beliebigen) deskriptiven Statistiken nach Subgruppen getrennt erstellen.

```
library(plyr)
dat <- data.frame(g = c(1, 1, 1, 1, 2, 2, 2, 2),
                  ## M_1 = 2, M_2 = 1
                  x = c(1, 2, 3, NA, 0.5, 1, 1.5, NA))
```

dat

	g	x
1	1	1.0
2	1	2.0
3	1	3.0
4	1	NA
5	2	0.5
6	2	1.0
7	2	1.5
8	2	NA

plyr: Deskriptive Statistiken II

```
countMiss <- function(x){return(sum(is.na(x)))}  
ddply(dat, .(g),  
  summarize,  
    m = mean(x, na.rm=TRUE), ## Mittelwert  
    sd = sd(x, na.rm=TRUE), ## Standardabweichung  
    n.miss = countMiss(x)) ## Anzahl missings
```

	g	m	sd	n.miss
1	1	2	1.0	1
2	2	1	0.5	1

plyr: Korrelationen I

Problem

Nach Gruppen getrennte Korrelationsmatrizen berechnen.

```
library(plyr)
set.seed(12)
dat <- data.frame(g = factor(rbinom(100, 1, 0.5),
                             labels = c("males", "females")),
                  x = rnorm(100),
                  y = rnorm(100),
                  z = rnorm(100))
head(dat)
```

	g	x	y	z
1	males	-0.04268	0.2523	-0.45486
2	females	-0.11267	0.6423	-0.08839
3	females	0.45683	-0.3469	0.03966
4	males	2.02033	-0.5153	0.54707
5	males	-1.05089	0.3982	-0.22569
6	males	0.73465	0.9536	0.39366

plyr: Korrelationen II

```
## Funktion zur Berechnung der Korr.matrix
## Die 1. Spalte mit der Gruppierungsvariablen ausschliessen
calcCorr <- function(x){
  cor(x[, -1])
}
## Eingabe: data.frame, Ausgabe: list
dply(dat, .(g), calcCorr)
```

\$males

	x	y	z
x	1.00000	-0.3309	-0.03331
y	-0.33089	1.0000	0.29752
z	-0.03331	0.2975	1.00000

\$females

	x	y	z
x	1.00000	0.16922	-0.05563
y	0.16922	1.00000	0.02364
z	-0.05563	0.02364	1.00000

```
attr("split_type")
[1] "data.frame"
attr("split_labels")
      g
1  males
2 females
```

Gliederung

Problem

Übersicht von Paketen und Befehlen

Statistische Funktionen (Aggregation)

Base R

Das `plyr`-Paket

Datenaufbereitung

No loops, please!

Weitere Quellen und Literatur

Group-mean centering I

Problem

In einem bestehenden Datensatz eine neue „group-mean centered“ Variable erstellen (etwa für Mehrebenenanalysen).

```
library(plyr)
dat <- data.frame(g = c(1, 1, 1, 2, 2, 2),
                  ## M_1 = 2, M_2 = 1
                  x = c(1, 2, 3, 0.5, 1, 1.5))
```

dat

	g	x
1	1	1.0
2	1	2.0
3	1	3.0
4	2	0.5
5	2	1.0
6	2	1.5

Group-mean centering II

```
ddply(dat, .(g), transform,  
       x.c = scale(x, scale = FALSE))
```

	g	x	x.c
1	1	1.0	-1.0
2	1	2.0	0.0
3	1	3.0	1.0
4	2	0.5	-0.5
5	2	1.0	0.0
6	2	1.5	0.5

Gliederung

Problem

Übersicht von Paketen und Befehlen

Statistische Funktionen (Aggregation)

Base R

Das `plyr`-Paket

Datenaufbereitung

No loops, please!

Weitere Quellen und Literatur

Das Beispielproblem

- Erzeuge n Datensätze mit vorgegebenem Mittelwert und Standardabweichung und speichere diese als Textdateien.
- Lade die n Textdateien (und lege die Datensätze in einem Listenobjekt ab).
- Berechne pro Datensatz verschiedene deskriptive Statistiken und speichere diese in einem neuen `data.frame`.
- Nachfolgend werden zwei Lösungsansätze vorgestellt, nämlich mit `lapply` und `plyr`.

Schleifen vermeiden mit lapply I

```
## Anzahl Datensätze
n <- 100
## data.frame-Objekt mit Parameterangaben und Dateinamen
means <- sample(0:20, n, replace = TRUE)
sd <- sample(1:10, n, replace = TRUE)
param <- data.frame(m = means, sd)
## Dateinamen
param$filename <- paste0("../..data/n_files/dat",
                          1:n, ".dat")
## Von data.frame zu list
param <- split(param, 1:nrow(param))
head(param, n = 2)

$`1`
  m sd filename
1 7  6 ../..data/n_files/dat1.dat

$`2`
  m sd filename
2 8  8 ../..data/n_files/dat2.dat
```


Schleifen vermeiden mit lapply II

```
## Funktion zum erzeugen der Datensätze & speichern  
genData <- function(x){  
  mySample <- rnorm(n, x$m, x$sd)  
  write.table(mySample, file = x$filename,  
              row.names = FALSE, col.names = FALSE)  
}
```

```
## Die "eigentliche Schleife"  
invisible(lapply(param, genData))  
## invisible() unterdrückt die Ausgabe...
```

```
## Jetzt die n Datenfiles wieder einlesen  
readData <- function(x){  
  dat <- read.table(file = x$filename)  
  return(dat)  
}
```

Schleifen vermeiden mit lapply III

```
## Die "eigentliche Schleife" zum Einlesen  
lData <- lapply(param, readData)
```

Ein Auszug aus dem Verzeichnis /n_files :

```
tmp <- list.files(path = "../..data/n_files")  
length(tmp)  
  
[1] 100  
  
head(tmp, n = 10)  
  
[1] "dat1.dat"    "dat10.dat"   "dat100.dat"  "dat11.dat"  
[5] "dat12.dat"   "dat13.dat"   "dat14.dat"   "dat15.dat"  
[9] "dat16.dat"   "dat17.dat"
```

Schleifen vermeiden mit lapply IV

```
## Einen data.frame mit deskript. Statistiken  
## erstellen  
calcDesc <- function(x){  
  m <- mean(x$V1)  
  sd <- sd(x$V1)  
  minimum <- min(x$V1)  
  maximum <- max(x$V1)  
  return(data.frame(m, sd, minimum, maximum))  
}  
lDesc <- lapply(lData, calcDesc)  
## Von list zu data.frame  
dDesc <- do.call("rbind", lDesc)  
head(dDesc)
```

	m	sd	minimum	maximum
1	11.184	8.968	-8.2830	31.53
2	16.582	8.028	-3.3826	36.54
3	5.191	6.783	-13.2454	19.84
4	16.168	7.375	0.6647	30.55
5	8.191	7.879	-6.7554	28.97
6	8.771	7.508	-7.0102	26.37

Schleifen vermeiden mit ddply I

```
## Anzahl Datensätze
n <- 100
## data.frame-Objekt mit Parameterangaben und Dateinamen
means <- sample(0:20, n, replace = TRUE)
sd <- sample(1:10, n, replace = TRUE)
param <- data.frame(m = means, sd)
param$filename <- paste0("../..data/n_files/dat",
                        1:n, ".dat")
head(param, n = 2)
```

	m	sd	filename
1	11	9	../..data/n_files/dat1.dat
2	17	8	../..data/n_files/dat2.dat

Schleifen vermeiden mit ddply II

```
## Funktion zum erzeugen der Datensätze & abspeichern  
genData <- function(x){  
  mySample <- rnorm(n, x$m, x$sd)  
  write.table(mySample, file = x$filename,  
              row.names = FALSE, col.names = FALSE)  
}
```

```
## Die "eigentliche Schleife"  
invisible(ddply(param, .(1:n), genData))  
## invisible() unterdrückt die Ausgabe...
```

```
## Jetzt die n Datenfiles wieder einlesen  
readData <- function(x){  
  dat <- read.table(file = x$filename)  
  return(dat)  
}
```

Schleifen vermeiden mit ddply III

```
## Die "eigentliche Schleife" zum Einlesen  
lData <- dply(param, .(1:n), readData)
```

Schleifen vermeiden mit ddply IV

```
## Einen data.frame mit deskript. Statistiken  
## erstellen  
calcDesc <- function(x){  
  m <- mean(x$V1)  
  sd <- sd(x$V1)  
  minimum <- min(x$V1)  
  maximum <- max(x$V1)  
  return(data.frame(m, sd, minimum, maximum))  
}  
## Von list zu data.frame  
dDesc1 <- ldply(lData, calcDesc)  
head(dDesc1)
```

	1:n	m	sd	minimum	maximum
1	1	11.184	8.968	-8.2830	31.53
2	2	16.582	8.028	-3.3826	36.54
3	3	5.191	6.783	-13.2454	19.84
4	4	16.168	7.375	0.6647	30.55
5	5	8.191	7.879	-6.7554	28.97
6	6	8.771	7.508	-7.0102	26.37

Gliederung

Problem

Übersicht von Paketen und Befehlen

Statistische Funktionen (Aggregation)

Base R

Das `plyr`-Paket

Datenaufbereitung





No loops, please!

Weitere Quellen und Literatur

Hilfreiche Quelle I

- Markus Gesmann hat sich sich bereits zu diesem Thema ausgelassen: „Say it in R with „by“, „apply“ and friends“ (Gesmann, 2013).
- Unbedingt Hadley Wickhams JSS-Artikel lesen (Wickham, 2011) bzw. die Website dazu besuchen: <http://plyr.had.co.nz/>.
- Eine schöne Darstellung aller `*pply`-Varianten findet sich hier: [A brief introduction to “apply” in R](#)
- Stata vs R in [Loops revisited: How to rethink macros when using R](#)
- Eine schöne Übersicht zu [Summarizing data](#)
- Auch auf CrossValidated wurde gefragt: [How to summarize data by group in R?](#)
- ... und auf Stackoverflow: [R Grouping functions: sapply vs. lapply vs. apply. vs. tapply vs. by vs. aggregate vs](#)

Literatur I

-  Gesmann, Markus (2013). *Say it in R with 'by', 'apply' and friends*. URL: <http://lamages.blogspot.de/2012/01/say-it-in-r-with-by-apply-and-friends.html> (besucht am 13.10.2013).
-  Matloff, Norman S (2011). *Art of R programming*. English. San Francisco, Calif.: No Starch Press. URL: <http://www.books24x7.com/marc.asp?bookid=44507> (besucht am 18.10.2013).
-  Shalizi, Cosma (2013). *Split, Apply, Combine: Using plyr (Introduction to Statistical Computing)*. URL: <http://vserver1.cscs.lsa.umich.edu/~crshalizi/weblog/1068.html> (besucht am 14.10.2013).
-  Teetor, Paul (2011). *R Cookbook*. O'Reilly Media. URL: <http://shop.oreilly.com/product/9780596809164.do> (besucht am 13.10.2013).

Literatur II



Wickham, Hadley (2011). „The Split-Apply-Combine Strategy for Data Analysis“. In: *Journal of Statistical Software* 40.1, S. 1–29. ISSN: 1548-7660. URL: <http://www.jstatsoft.org/v40/i01>.