# Datenmanipulation mit PLYR, DPLYR und reshape2

Hung Quan Vu

Fachbereich Informatik

Betreuer: Jakob Lüttgau

# Inhalt:

# PLYR

a. Was ist PLYR ?

- Plyr ist ein R-Paket das vereinfacht den Dateisaufteilungsvorgang, Dateisbearbeitungsvorgang und Dateisvereinigungsvorgang. Dies sind gemeinsame Datenmanipulationsschritt. Wichtig ist, dass plyr es leicht macht, die Eingangs- und Ausgangsdatenformat von einem syntaktisch konsistenten Satz von Funktionen zu steuern.

- Zb plyr ist sehr effektiv wenn wir wollen:

  - passen das gleiche Modell für jeder subsets eines Dataframe
  - zusammenpassende Statistiken für jede Gruppe schnell berechnen
  - Führen gruppenweise Transformationen wie Skalieren oder Standardisierung

- Es ist bereits möglich, dies mit der Basis R-Funktionen zu Machen (wie split und apply- Funktionsfamilie, aber mit plyr ist es einbisschen leichter mit:

  - völlig konsistente Namen, Argumente und Ausgänge
  - bequem Parallelisierung durch die foreach-Paket
  - Eingabe von und Ausgabe an data.frames, Matrizen und Listen
  - Fortschrittsbalken für lang laufenden Betrieb zu verfolgen
  - eingebaute Fehlerwiederherstellung und informative Fehlermeldungen
  - Etiketten, die für alle Transformationen beibehalten werden

b. Warum wollen wir PLYR benutzen?

- PLYR > Basis Apply Funktion > Schleife (for)
  - Apply > Schleife
    +) Die Kodierung ist saubere (wenn man mit dem Konzept sich vertraut sind). Es ist nicht nur einfacher zu codieren und zu lesen, sondern auch weniger fehleranfällig, weil:
    (a) Man muss sich nicht mit “Subsetting” beschäftigen
    (b) Man muss sich nicht mit dem Speichern der Ergebnisse beschäftigen
    +) Apply-Funktionen kann schneller als Schleifen sein, manchmal sogar dramatisch.
  - PLYR > Basis Aplly-Funktionen
    +) plyr hat eine gemeinsame Syntax – leichter

+) plyr erfordert weniger Code, da es sich um der Eingabe – und Ausgabeformat kümmert.
+) plyr kann leicht parallel betrieben werden -> schneller

c. PLYR basics

- Plyr wurde auf dem Integrierten in Anwendungsfunktionen gebaut, indem man den Ein- und Ausgabeformate manipulieren kann und die Syntax konsistent in allen Variationen zu halten. Es fügt auch einige Feinheiten wie Fehlerverarbeitung, parallelverarbeitung und Fortschrittsbalken.
- Das Basisformat ist 2 Buchstaben und dann ply(). Der erste Buchstabe bezieht sich auf das Eingabesformat und die zweite auf das Ausgabesformat.

- Die 3 Hauptbuchstaben sind:
  - d = data frame
  - a = array (enthält Matrizen)
  - l = liste

| | dataframe | list | array |
|---|---|---|---|
| dataframe | ddply | ldply | adply |
| List | dlply | llply | alply |
| array | daply | laply | aaply |

- Einige weniger häufig Buchstaben: m, r , _
  - m = multi-Argument Funktionseingang
  - r = eine Funktion n-mal wiederholen.
  - _ = Ausgabe wegwerfen

Zum plotting könnte man den Unterstrich (_) nutzlich finden. Es wird etwas mit den Daten (zb hinzufügen Liniensegmente zu einem Grundstück) zu tun und dann die Ausgabe wegzuwerfen( zb d_ply()).

d. Ein allgemeines Beispiel mit plyr

Nehmen wir ein einfaches Beispiel: eine Datenframe, “split” die(durch Jahre), die Koffizient von Variation der Zählung berechnen, und dann eine Datenframe zurückliefern.

```
> set.seed(1)
> d <- data.frame(year = rep(2000:2002, each = 3),+ count = round(runif(9, 0, 20)))
> print(d)
```

| | Year | Count |
|---|---|---|
| 1 | 2000 | 5 |
| 2 | 2000 | 7 |
| 3 | 2000 | 11 |
| 4 | 2001 | 18 |
| 5 | 2001 | 4 |
| 6 | 2001 | 18 |
| 7 | 2002 | 19 |
| 8 | 2002 | 13 |
| 9 | 2002 | 13 |

```
> library(plyr)
> ddply(d, "year", function(x) {+ mean.count <- mean(x$count)+ sd.count <- sd(x$count)+ cv <- sd.count/mean.count
+ data.frame(cv.count = cv)
+ })
```

| | year | cv.count |
|---|---|---|
| 1 | 2000 | 0.3984848 |
| 2 | 2001 | 0.6062178 |
| 3 | 2002 | 0.2309401 |

e. Umwandeln und zusammenfassen (transform and summarise)
   - Es ist oft bequen, diese Funktionen innerhalb plyr zu verwenden.
   - Transformation wirkt als es wäre eine ganze normale Basisfunktion von R und ändert eine bestehende Datenframe.
   - Summarise erstellt eine neue (in der Regel) verkürzte Datenframe

```
> ddply(d, "year", summarise, mean.count = mean(count))
```

| | year | mean.count |
|---|---|---|
| 1 | 2000 | 7.666667 |
| 2 | 2001 | 13.333333 |
| 3 | 2002 | 15.000000 |

```
> ddply(d, "year", transform, total.count = sum(count))
```

| | Year | Count | Total count |
|---|---|---|---|
| 1 | 2000 | 5 | 23 |
| 2 | 2000 | 7 | 23 |
| 3 | 2000 | 11 | 23 |
| 4 | 2001 | 18 | 40 |
| 5 | 2001 | 4 | 40 |
| 6 | 2001 | 18 | 40 |
| 7 | 2002 | 19 | 45 |
| 8 | 2002 | 13 | 45 |
| 9 | 2002 | 13 | 45 |

Bonus Funktion: mutate. mutate funktioniert fast genau wie transform aber die ermöglich es, Spalten aus Spalten die von Nutzer erzeugt werden bauen.

```
> ddply(d, "year", mutate, mu = mean(count), sigma = sd(count),
+ cv = sigma/mu)
```

| | Year | Count | mu | sigma | cv |
|---|---|---|---|---|---|
| 1 | 2000 | 5 | 7.666667 | 3.055050 | 0.3984848 |
| 2 | 2000 | 7 | 7.666667 | 3.055050 | 0.3984848 |
| 3 | 2000 | 11 | 7.666667 | 3.055050 | 0.3984848 |
| 4 | 2001 | 18 | 13.333333 | 8.082904 | 0.6062178 |
| 5 | 2001 | 4 | 13.333333 | 8.082904 | 0.6062178 |
| 6 | 2001 | 18 | 13.333333 | 8.082904 | 0.6062178 |
| 7 | 2002 | 19 | 15.000000 | 3.464102 | 0.2309401 |
| 8 | 2002 | 13 | 15.000000 | 3.464102 | 0.2309401 |
| 9 | 2002 | 13 | 15.000000 | 3.464102 | 0.2309401 |

f. Nested chunking of the data
   - Die grundlegende Syntax können leicht erweitert werden, um die Daten auseinander zu brechen basierend auf mehreren Spalten

```
> baseball.dat <- subset(baseball, year > 2000) # data from the plyr package
> x <- ddply(baseball.dat, c("year", "team"), summarize,
+ homeruns = sum(hr))
> head(x)
```

| | year | Team | homeruns |
|---|---|---|---|
| 1 | 2001 | ANA | 4 |
| 2 | 2001 | ARI | 155 |
| 3 | 2001 | ATL | 63 |
| 4 | 2001 | BAL | 58 |
| 5 | 2001 | BOS | 77 |
| 6 | 2001 | CHA | 63 |

g. Andere nutzliche Möglichkeiten

- Fehler behandeln: man kann die failwith Funktion benutzen , umzu steuern , wie Fehler behandelt werden.

```
> f <- function(x) if (x == 1) stop("Error!") else 1
> safe.f <- failwith(NA, f, quiet = TRUE)
> #llply(1:2, f)
> llply(1:2, safe.f)

[[1]]
[1] NA

[[2]]
[1] 1
```

- Parallelverarbeitung

  In Verbindung mit domc (oder doSMP unter Windows) kann man separate Funktionen auf jedem Kern von dem Computer ausführen. Auf einem Dual-Core-Maschine kann die Geschwindigkeit in einigen Situationen verdoppeln. Set .Parallel = TRUE.

```
> x <- c(1:10)
> wait <- function(i) Sys.sleep(0.1)
> system.time(llply(x, wait))

user   system elapsed
0.001  0.000    1.007

> system.time(sapply(x, wait))



user     system elapsed
0.001    0.000    1.010

> library(doMC)
> registerDoMC(2)
> system.time(llply(x, wait, .parallel = TRUE))

user    system elapsed
0.021   0.006    0.533
```

h. Warum will ich PLYR nicht benutzen ?

Plyr kann langsam sein. Vor allem wenn man mit sehr großen Datenmengen arbeitet, die beinhalten eine Menge subsetting. Hadley arbeitet an dieser Stelle und die letzte Entwicklungsversionen von plyr kann schon viel schneller laufen.

Es gibt 3 schneller Möglichkeiten:

- (1) Basis R-apply-Funktion verwenden:

```
> system.time(ddply(baseball, "id", summarize, length(year)))

User     system elapsed
1.169    0.013     1.188

> system.time(tapply(baseball$year, baseball$id, function(x) length(x)))

User    system elapsed
0.019    0.000    0.020
```

- (2) Immutable data frame verwenden. Eine immutable data frame (idata.frame) liefert bei subseting den Zeigers auf das ursprüngliche Objekt zurück, staat eine Kopie von sich erstellen. Dies ist oft der limitierende Schritt in einer apply-Funktion.

```
> system.time(ddply(idata.frame(baseball), "id", summarize, length(year)))

User   system elapsed
0.956   0.003    0.960
```

- (3) Data.table Paket verwenden:

```
> library(data.table)
> dt <- data.table(baseball, key="id")
> system.time(dt[, length(year), by=list(id)])

user   system elapsed
0.005  0.000   0.005
```

# DPLYR

a. Was ist DPLYR

- DPLYR ist die nächste Iteration von PLYR
- Ziel ist nur Data frames
- DPLYR ist schneller als PLYR, hat mehr konsistent API

b. Warum wollen wir DPYR benutzen?

- Zeit ist Geld , deswegen Romain Francois hat die wichtigsten Stücke in RCPP geschrieben, um schnelle Leistung zu prallen. Die Leistung wird in der Laufe der Zeit nur besser werden, besondert wenn wir den besten Weg herrausfinden, die meisten von mehreren Prozessoren zu nutzen( 20 – 1000x schneller als plyr)

- "Tabular data is tabular data regardless of where it lives, so you should use the same functions to work with it". In dplyr alles was man mit lokale Datenframe tun kann man auch mit Remote-Datenbanktabelle machen. PostgreSQL, MySQL, SQLite und Google bigquery haben built-in-support; neue Backend hinzufügen ist eigentlich S3-Methode zu implementieren.

- Der Engpass in den moisten Datenanalysen ist die Zeit, die man braucht, um herrauszufinden was man mit seinen Daten tun, und dplyr macht dies einfacher durch einzehlne Funktionen aufzuweisen. (group_by, summarise, mutate, filter, select und arrange). Jede Funktion löse nur eine Aufgabe, aber der tut es wirklich effizient.

- Verkettungs Befehle mit %.% (%>%)

c. Die Basis

- Manipuliertsfunktionen
    - filter: subset Zeilen. Mehrere Bedingungen kombiniert von & nur; | ist nicht verfügbar.
    - select : subset Spalten. Mehrere Spalten könne zurückgegeben werden.
    - arrange: Zeilen umordnen. Bietet Platz für mehrere Eingänge und auf- / absteigend Ordnung.
    - mutate: fügt neue Spalten hinzu, möglicherweise auf der Grundlage anderer Spalten; mehrere Eingänge erstellen mehrere Spalten.
    - summarize: jede Funktion innerhalb der Gruppen zu berechnen, so dass jede Gruppe auf eine einzige Zeile reduziert. Mehrere Eingänge erstellen mehrere Ausgabe summarize.

- Chaining commands
    - Nested R-Befehle sind oft schwierig zu lesen
        - die Reihenfolge der Operationen aus dem innersten zu den äußersten Funktionen zu lesen.
        - Folglich treten die Argumente für diese äußerste Funktionen ein langer Weg weg von der eigentlichen Funktion.
        - dplyr ermöglicht es , Befehlen mit der Kette oder %.% Funktionen der Operationen linear zu sequenzieren, und damit viel mehr logisch.

- Datenbank
    - In Anlehnung an Hadley, unterstützt dplyr die drei populärsten Open-Source-Datenbanken (SQLite, MySQL und PostgreSQL) und Google BigQuery.
    - Nutzlich wenn wir unsere Daten aus der Datenbank nicht extrahieren wollen.

# Reshape2

a. Was ist Reshape2

- Ein R Paket die vereinfacht Dateisumwandlungsvorgang zwischen long und wide Format.
- Ein Neustart des reshape Paket
- Datensumformung deutlich stärker konzentiert und viel schneller
- verbessert die Geschwindigkeit auf das Kosten der Funktionalität

b. Verbesserung

- wesentlich schneller und mehr Speicher effizient
- cast wird durch zwei Funktionen in Abhängigkeit vom Ausgangstyp ersetzt: dcast erzeugt dataframe und acast erzeugt Matrizen / Arrays.
- mehrdimensionale Margen sind jetzt möglich
- Einige Funktionen wurden wie die entfernt zb. der| cast-Operator, und die Möglichkeit, mehrere Werte von einer Aggregationsfunktion zurückzuliefern.
- eine bessere Entwicklungspraktiken wie Namespaces und Tests.

c. Was ist wide und long Datei ?

Wide-daten hat eine Spalte für jede Variable. Zb:

```
#  ozone  wind  temp
# 1 23.62 11.623 65.55
# 2 29.44 10.267 79.10
# 3 59.12  8.942 83.90
# 4 59.96  8.794 83.97
```

Und hier ist long-format Daten:

```
#    variable  value
# 1     ozone 23.615
# 2     ozone 29.444
# 3     ozone 59.115
# 4     ozone 59.962
# 5      wind 11.623
# 6      wind 10.267
# 7      wind  8.942
# 8      wind  8.794
# 9      temp 65.548
# 10     temp 79.100
# 11     temp 83.903
# 12     temp 83.968
```

- Long-format-daten hat eine Spalte für möglichen Variablentypen und eine Spalte für die Werte dieser Variablen. Long-format-daten ist nicht notwendigerweise nur zwei Spalten. Zum Beispiel könnten wir Ozonmessungen für jeden Tag des Jahres haben. In diesem Fall könnte man eine andere Spalte für Tage. Mit anderen Worten, gibt es verschiedene Ebenen von „Longness“. Die ultimative Form die wir wollen um unsere Daten erhalten hängt davon ab, was wir mit ihm tun werden.

- Es stellt sich heraus, dass man wide-format-daten für einige Arten von Datenanalyse und long-format-daten für die andere benötigen. In Wirklichkeit brauchen wir viel mehr long-format-daten als wide-format-daten. Zum Beispiel ggplot2 long-format-daten ( technische saubere Daten) erforderlich ist. Plyr erfordert auch long-format-daten, und die meisten Modellierungsfunktionen ( wie lm(), glm(), and gam()) erfordern long-format-daten. Aber die meisten Menschen finden es oft leichter, ihre Daten im wide-format-daten aufzuzeichnen.

d. Die Pakete

- melt umwandelt wide -format nach long-format

- cast umwandelt long -format nach wide – format

- “Think of working with metal: if you melt metal, it drips and becomes long. If you cast it into a mould, it becomes wide.”-Sean Anderson

# Zusammenfassung:

- plyr
  - Ist ein R Paket das vereinfacht den Dateisaufteilungsvorgang, Dateisbearbeitungsvorgang und Dateisvereinigungsvorgang.
  - Die Basisformatte ist 2 Buchstaben und dann ply().
  - Ist nicht immer schnell

- dplyr
  - ist die nächste Iteration von PLYR
  - Ziel ist nur Data frames
  - ist schneller als PLYR, hat mehr konsistent API
  - Manipuliertsfunktionen: filter, arrange, mutate, select, summarize
  - Channing command: dplyr ermöglicht es , Befehlen mit der Kette oder %.% Funktionen der Operationen linear zu sequenzieren, und damit viel mehr logisch als Nested R-Befehle.

- reshape2
  - Ein R Paket die vereinfacht Dateisumwandlungsvorgang zwischen long und wide Format.
  - Ein Neustart des reshape Paket
  - Basisfunktionen:
    - melt umwandelt wide -format nach long-format
    - cast umwandelt long -format nach wide -format

# Literatur


- https://cran.r-project.org
- https://inside-r.org
- https://rdocumentation.org
- https://r-bloggers.com
- The Comprehensive R Archive Network