Daten transformieren mit dplyr : : SCHUMMELZETTEL

dplyr Funktionen sind mit Pipes (alias Verkettungen) kompartibel und erwarten aufgeräumte Daten. Aufgeräumte Daten sind:

Jede **Variable** ist in einer eigenen **Spalte**

&

Jede **Beobachtung** (jeder **Datensatz**) ist in einer eigenen **Zeile**

Pipes (oder Verkettungen)

x %>% f(y)
wird zu **f(x, y)**

Datensätze zusammenfassen

Zusammenfassungs-Funktionen werden auf Spalten angewendet und erstellen eine neue Tabelle. Sie sind Funktionen, die einen Vektor als Eingabe haben und einen einzelnen Ausgabewert haben (siehe nächste Seite).

Zusammenfassungs-Funktion

summarise(.data, …)
Tabelle mit Summen berechnen. Ebenso **summarise_**().
summarise(mtcars, avg = mean(mpg))

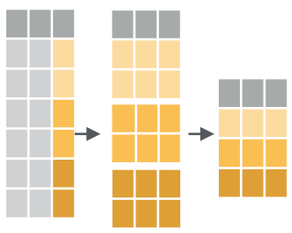
count(x, …, wt = NULL, sort = FALSE)
Anzahl der Zeilen mit jedem eindeutigen Wert der Variablen zählen, gruppiert nach den Variablen in … . Ebenso **tally**().
count(iris, Species)

VARIATIONEN

summarise_all() - Auf jede Spalte anwenden.
summarise_at() - Auf bestimmte Spalten anwenden.
summarise_if() - Auf Spalten eines Typus anwenden.

Datensätze gruppieren

Mit **group_by()** wird eine neue „gruppierte" Tabelle erstellt. dplyr Funktionen manipulieren jede „Gruppe" getrennt und kombinieren die Resultate.



```
mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))
```

group_by(.data, ..., add = FALSE**)**
Kopie einer Tabelle, gruppiert nach den Variablen in … .
g_iris <- group_by(iris, Species)

ungroup(x, …**)**
Kopie einer Tabelle, mit aufgehobenen Gruppierungen.
ungroup(g_iris)

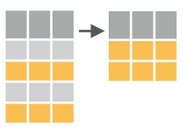
Datensätze manipulieren

ZEILEN EXTRAHIEREN

Mit einer Untermenge der Zeilen wird eine neue Tabelle erstellt. Für nicht-Standard-evaluierenden Code ist eine Variante mit Endung _ zu verwenden.

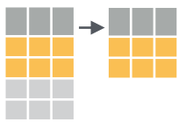
filter(.data, …**)**
Zeilen extrahieren die eine Bedingung erfüllen. Ebenso **filter_()**.
filter(iris, Sepal.Length > 7)

distinct(.data, ..., .keep_all = FALSE**)**
Duplikate entfernen (zeilenweise). Ebenso **distinct_()**.
distinct(iris, Species)



sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()**)**
Bruchteil der Zeilen stichprobenartig auswählen.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()**)**
n Zeilen stichprobenartig auswählen.
sample_n(iris, 10, replace = TRUE)



slice(.data, …**)**
Zeilen anhand ihrer Position auswählen. Ebenso **slice_()**.
slice(iris, 10:15)

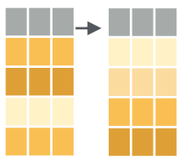
top_n(x, n, wt**)** Beste n Einträge auswählen und sortieren (nach Gruppe falls die Daten gruppiert sind).
top_n(iris, 5, Sepal.Width)

Logische und boolsche Operatoren mit filter() verwendbar

<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

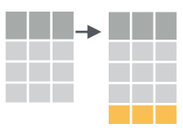
Siehe **?base::logic** und **?Comparison** für Hilfe.

ZEILEN ORDNEN



arrange(.data, …**)**
Zeilen anhand von Werten in einer Spalte sortieren (von klein nach groß). Mit **desc()** kann die Sortierung umgedreht werden.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

NEUE ZEILEN HINZUFÜGEN



add_row(.data, ..., .before = NULL, .after = NULL**)**
Eine oder mehrere Zeilen hinzufügen.
add_row(faithful, eruptions = 1, waiting = 1)

Variablen manipulieren

VARIABLEN EXTRAHIEREN

Mit einer Untermenge der Spalten wird eine neue Tabelle erstellt. Für nicht-Standard-evaluierenden Code ist eine Variante mit Endung _ zu verwenden.

select(.data, …**)**
Spalten anhand ihres Namens auswählen. Ebenso **select_if()**
select(iris, Sepal.Length, Species)

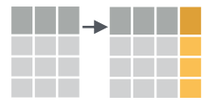
Hilfsfunktionen für select ()
z.B. select(iris, starts_with("Sepal"))

contains(match**)** **num_range(**prefix, range**)** **:** z.B. mpg:cyl
ends_with(match**)** **one_of(**…**)** **-** z.B. -Species
matches(match**)** **starts_with(**match**)**

NEUE VARIABLEN ERSTELLEN

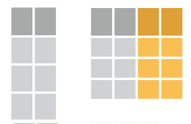
Fenster-Funktionen werden auf Spalten angewendet. Sie sind Funktionen, die einen Vektor als Eingabe und (mit gleicher Länge) als Ausgabe haben (siehe nächste Seite).

Fenster-Funktion



mutate(.data, …**)**
Neue Spalten berechnen und hinzufügen.
mutate(mtcars, gpm = 1/mpg)

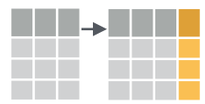
transmute(.data, …**)**
Neue Spalten berechnen und ursprüngliche Spalten entfernen.
transmute(mtcars, gpm = 1/mpg)



mutate_all(.tbl, .funs, …**)** Auf jede Spalte anwenden. Verwendbar mit **funs()**.
mutate_all(faithful, funs(log(.), log2(.)))

mutate_at(.tbl, .cols, .funs, …**)** Auf bestimmte Spalten anwenden. Verwendbar mit **funs()**, **vars()** und den Hilfsfunktionen für select().
mutate_at(iris, vars(-Species), funs(log(.)))

mutate_if(.tbl, .predicate, .funs, …**)**
Auf jede Spalte eines Typus anwenden. Verwendbar mit **funs()**.
mutate_if(iris, is.numeric, funs(log(.)))



add_column(.data, …, .before = NULL, .after = NULL**)**
Neue Spalte hinzufügen.
add_column(mtcars, new = 1:32)

rename(.data, …**)** Spalte umbenennen.
rename(iris, Length = Sepal.Length)

Fenster-Funktionen

MIT MUTATE() ZU VERWENDEN

mutate() und **transmute()** wenden vektorisierte Funktionen auf Spalten an um neue Spalten zu erstellen. Diese sog. Fenster-Funktionen haben einen Vektor als Eingabe und einen Vektor gleicher Länge als Ausgabe.

Fenster-Funktion

OFFSETS

dplyr::**lag()** Werteverschiebung um 1 nach hinten
dplyr::**lead()** Werteverschiebung um 1 nach vorne

KUMULATIVE AGGREGIERUNGEN

dplyr::**cumall()** Kumulatives all()
dplyr::**cumany()** Kumulatives any()
cummax() Kumulatives max()
dplyr::**cummean()** Kumulatives mean()
cummin() Kumulatives min()
cumprod() Kumulatives prod()
cumsum() Kumulatives sum()

RANKINGS

dplyr::**cume_dist()** Summenverteilung als Proportion aller Werte <=
dplyr::**dense_rank()** Rangordnung ohne Lücke, mit **min** zur Unentschieden-auflösung bei Gleichstand
dplyr::**min_rank()** Rang mit **min** bei Gleichstand
dplyr::**ntile()** Einteilung in n Klassen
dplyr::**percent_rank()** min_rank auf [0,1] skaliert
dplyr::**row_number()** Rang mit "**first**" bei Gleichst.

MATHEMATIK

+, - , *, /, ^, %/%, %% arithmetische Operanden
log(), log2(), log10() Logarithmen
<, <=, >, >=, !=, == logische Vergleiche

DIVERSE

dplyr::**between()** x >= links & x <= rechts
dplyr::**case_when()** mehrfaches **if_else()**
dplyr::**coalesce()** elementweiser erster nicht-NA Wert, angewendet auf Vektoren
dplyr::**if_else()** elementweises **if()** + **else()**
dplyr::**na_if()** bestimmte Werte durch NA ersetzen
pmax() elementweises **max()**
pmin() elementweises **min()**
dplyr::**recode()** vekotirisiertes **switch()**
dplyr::**recode_factor()** vektorisiertes **switch()** für Faktoren

Zusammenfassungs-Fkt.

MIT SUMMARISE() ZU VERWENDEN

summarise() wendet Zusammenfassungs-Funktionen auf Spalten an um eine neue Tabelle zu erstellen. Sie haben einen Vektor als Eingabe und einen einzelnen Wert aus Ausgabe.

Zusammenfassungs-Funktion

ZÄHLUNG

dplyr::**n()** Anzahl Zeilen
dplyr::**n_distinct()** Anzahl eindeutiger Datensätze
sum(!is.na()) Anzahl von nicht-NAs

LOKATION

mean() arithmetisches Mittel, ebenso **mean(!is.na())**
median() Median

BOOLEAN

mean() Anteil der TRUEs
sum() Anzahl TRUEs

POSITION/ORDNUNG

dplyr::**first()** erster Wert eines Vektors
dplyr::**last()** letzter Wert eines Vektors
dplyr::**nth()** n-ter Wert eines Vektors

RANG

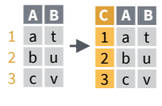
quantile() n-tes Quantil
min() kleinster Wert
max() größter Wert

STREUUNG

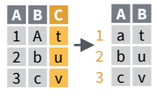
IQR() Interquartilsabstand eines Vektors
mad() mittlere absolute Abweichung
sd() Standardabweichung
var() Varianz

Zeilennamen

Aufgeräumte Daten verwenden keine Zeilennamen (diese wären außerhalb der Spalten gespeichert). Um mit Zeilennamen zu arbeiten, sind diese in eine Spalte einzufügen.



rownames_to_column()
Zeilennamen in neue Spalte verschieben
a <- rownames_to_column(iris, var = "C")

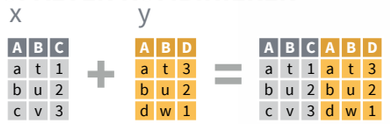


column_to_rownames()
Spalte in Zeilennamen verschieben
column_to_rownames(a, var = "C")

Ebenso **has_rownames()**, **remove_rownames()**

Zusammenfassungs-Fkt.

SPALTEN KOMBINIEREN



Mit **bind_cols()** werden zwei Tabellen so wie sie sind nebeneinander zusammengefügt.

bind_cols(…) erstellt eine neue Ausgabe-Tabelle als nebeneinander gestellte Eingabe-Tabellen.
WICHTIG: VORAB SICHERGEHEN, DASS DIE ZEILEN GLEICH AUSGERICHTET SIND.

Mit einem "v**eränderndеn Join**" wird eine Tabelle zu Spalten einer anderen Tabelle hinzugefügt, basierend auf identischen Werten in den Zeilen. Jeder Join bewahrt eine andere Kombination der Werte aus den Tabellen.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…**)**
Übereinstimmende Werte von y zu x anfügen

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…**)**
Übereinstimmende Werte von x zu y anfügen

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…**)**
Daten vereinigen, nur Zeilen mit beiderseitigen Übereinstimmungen werden behalten

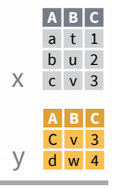
full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…**)**
Daten vereinigen , alle Zeilen werden behalten

Mit **by = c("col1", "col2")** werden die Spalte(n) für die Übereinstimmungen bestimmt
left_join(x, y, by = "A")

Mit einem benannten Vektor, **by = c("col1" = "col2")**, können Spalten mit unterschiedlichen Namen in den Tabellen verglichen werden.
left_join(x, y, by = c("C" = "D"))

Mit **suffix** wird ein Suffix für gleichnamige Tabellenspalten bestimmt, um duplizierte Spaltennamen zu vermeiden.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

ZEILEN KOMBINIEREN



Mit **bind_rows()** werden Tabellen so wie sie sind untereinander angefügt.

bind_rows(…, .id = NULL**)** erstellt eine neue Tabelle als untereinandergestellte Eingabe-Tabellen. Um eine neue Spalte mit den jeweiligen Tabellennamen hinzuzufügen, bekommt .id einen Spaltennamen zugewiesen (siehe Grafik links).

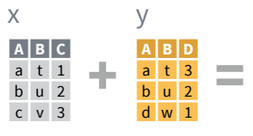
intersect(x, y, …)
Schnittmenge, d. h. Zeilen die in **x** und **y** vorkommen

setdiff(x, y, …)
Differenzmenge, d. h. Zeilen von **x** die nicht in **y** vorkommen

union(x, y, …)
Vereinigungsmenge, d. h. Zeilen die in einem oder beiden vorkommen (ohne Duplikate). **union_all()** behält Duplikate.

Mit **setequal()** kann getestet werden, ob zwei Tabellen die exakt gleichen Zeilen beinhalten (die Reihenfolge ist egal).

ZEILEN EXTRAHIEREN



Mit einem "**filternden Join**" wird eine Tabelle anhand der Zeilen einer anderen Tabelle gefiltert.

semi_join(x, y, by = NULL, …**)**
Alle Zeilen von **x** mit Übereinstimmung in **y**.
NÜTZLICH UM ZU SEHEN,
WAS VERBUNDEN WIRD.

anti_join(x, y, by = NULL, …**)**
Alle Zeilen von **x** ohne Übereinstimmung in **y**.
NÜTZLICH UM ZU SEHEN,
WAS NICHT VERBUNDEN WIRD.

Weitere Informationen auf browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.5.0 • tibble 1.2.0 • Update: 2017-01