

R-Grundlagen und Data Wrangling

Hilfe aufrufen

`help(topic)` oder `?topic` Dokumentation zu *topic*.
`help.search("topic")` Durchsucht die Hilfe nach *topic*.
`apropos("topic")` Findet Objekte im Suchpfad, deren Name (teilweise) dem regulären Ausdruck *topic* entspricht.
`help.start()` Öffnet die HTML-Version der Hilfe.

Arbeitsverzeichnis

`getwd()` Gibt das aktuelle Arbeitsverzeichnis aus.
`setwd("path/to/files")` Setzt das Arbeitsverzeichnis.
`dir()` Zeigt die Dateien im aktuellen Verzeichnis an (identisch mit `list.files()`).
`dir(recursive=TRUE)` Zeigt alle Dateien einschließlich der Unterverzeichnisse an.

Arbeitsumgebung

`ls()` Listet alle Variablen der Umgebung auf.
`rm(x)` Entfernt *x* aus der Umgebung.
`rm(list = ls())` Entfernt alle Variablen aus der Umgebung.

Pakete

`install.packages()` Installiert Pakete aus Repositories oder lokalen Dateien.
`library(dplyr)` Lädt das Zusatzpaket 'dplyr' und hängt es an den Suchpfad an.
`require(readr)` Lädt das Zusatzpaket 'readr' und hängt es an; gibt *FALSE* mit einer Warnung zurück (statt eines Fehlers), falls das Paket fehlt.
`search()` Gibt die Liste der angehängten Pakete aus.
`attach(name)` Hängt eine Menge von R-Objekten an den Suchpfad an (mit Vorsicht verwenden).
`detach("package:name")` Entfernt Objekte oder Pakete aus dem Suchpfad.
`vignette("dplyr")` Zeigt eine bestimmte Paket-Vignette an (hier von 'dplyr').
`browseVignettes("dplyr")` Listet die verfügbaren Vignetten des Pakets 'dplyr' im HTML-Browser auf.
`dplyr::filter()` Verwendet eine bestimmte Funktion aus einem Paket (hier filter aus 'dplyr').

Operatoren

Zuweisung

<code><-</code>	Weist den rechten Wert dem linken Namen zu.	<code><<-</code>	Linksseitig, weist in der übergeordneten Umgebung zu (innerhalb von Funktionen).
<code>=</code>	Wie <code><-</code> ; nicht für Zuweisungen innerhalb von Funktionsaufrufen verwendbar.	<code>--></code>	Rechtsseitig, weist in der übergeordneten Umgebung zu.
<code>-></code>	Weist den linken Wert dem rechten Namen zu.		

Arithmetik

<code>+</code>	Addition.
<code>-</code>	Subtraktion.
<code>*</code>	Multiplikation.
<code>/</code>	Division.
<code>^</code>	Potenzierung.
<code>x%%y</code>	Modulo (x mod y); 5%%2 ergibt 1.
<code>x%/y</code>	Ganzzahldivision; 5/%2 ergibt 2.

Vergleich

<code><</code>	Kleiner als.
<code><=</code>	Kleiner als oder gleich.
<code>></code>	Größer als.
<code>>=</code>	Größer als oder gleich.
<code>==</code>	Exakt gleich.
<code>!=</code>	Ungleich.

Logik

<code>&</code>	Elementweises UND.
<code> </code>	Elementweises ODER.
<code>!</code>	Elementweises NICHT.
<code>&&</code>	Logisches UND für Einzelwerte (Kurzschlussauswertung; in <code>if()</code> verwenden).
<code> </code>	Logisches ODER für Einzelwerte (Kurzschlussauswertung; in <code>if()</code> verwenden).

Sonstige

<code>:</code>	Erzeugt eine reguläre Sequenz.
<code>%in%</code>	Abgleich von Werten.
<code>%*%</code>	Matrixmultiplikation.
<code> ></code>	Base-R-Pipe ($R \geq 4.1$).
<code>%>%</code>	Pipe-Operator des Pakets <i>magrittr</i> .

Datentypen und Konvertierung

Übersicht

Datentyp	Beispiel	Prüfen	Konvertieren
logical	<code>x <- c(TRUE, FALSE)</code>	<code>is.logical(x)</code>	<code>as.logical(x)</code>
integer	<code>x <- c(5L, 9L)</code>	<code>is.integer(x)</code>	<code>as.integer(x)</code>
double	<code>x <- c(1.5, 4.7)</code>	<code>is.double(x)</code>	<code>as.double(x)</code>
numeric	<code>x <- c(5L, 4.7)</code>	<code>is.numeric(x)</code>	<code>as.numeric(x)</code>
complex	<code>x <- c(5i, 9i)</code>	<code>is.complex(x)</code>	<code>as.complex(x)</code>
character	<code>x <- c("A", "B")</code>	<code>is.character(x)</code>	<code>as.character(x)</code>
NULL	<code>x <- NULL</code>	<code>is.null(x)</code>	<code>as.null(x)</code>
NA	<code>x <- c(NA, NA)</code>	<code>is.na(x)</code>	
factor	<code>x <- factor(c("A", "B"))</code>	<code>is.factor(x)</code>	<code>as.factor(x)</code>
date	<code>x <- as.Date("2020-06-01")</code>	<code>inherits(x, "Date")</code>	<code>as.Date(x)</code>

Eine vollständige Liste der Prüf- und Konvertierungsfunktionen liefern `methods(is)` und `methods(as)`.

`typeof(x)` Bestimmt den (R-internen) Typ bzw. Speichermodus eines Objekts.

Funktionen für logische Datentypen

`isTRUE(x)` Prüft, ob `x` ein einzelner TRUE-Wert ist.

`isFALSE(x)` Prüft, ob `x` ein einzelner FALSE-Wert ist.

`any(x)` Prüft, ob mindestens ein Wert in `x` TRUE ist.

`all(x)` Prüft, ob alle Werte in `x` TRUE sind.

Datenobjekte in R

Homogene Datentypen

			
	Vektor	Matrix	Array
Erstellen	<code>c(), rep(x, ...), seq(from, to, by)</code>	<code>matrix(data, nrow, ncol), cbind(), rbind()</code>	<code>array(data, dim)</code>
Indizieren	<code>[...]</code>	<code>[...]</code>	<code>[..., ..., ...]</code>
Prüfen	<code>is.vector(x)</code>	<code>is.matrix(x)</code>	<code>is.array(x)</code>
Konvertieren	<code>as.vector(x)</code>	<code>as.matrix(x)</code>	<code>as.array(x)</code>
Dimension <code>dim(x)</code>	1 (NULL)	2 (Zeilen, Spalten)	3 (Zeilen, Spalten, Ebenen)
Länge <code>length(x)</code>	Elemente im Vektor	Zeilen*Spalten	Zeilen*Spalten*Ebenen
Namen	<code>names(x)</code>	<code>colnames(x), rownames(x)</code>	<code>dimnames(x)</code>

Heterogene Datentypen

			
	Liste	Data Frame	Tibble
Erstellen	<code>list()</code>	<code>data.frame()</code>	<code>tibble::tibble()</code>
Indizieren	<code>\$</code> oder <code>[...]</code>	<code>\$</code> oder <code>[...] oder [...]</code>	<code>\$</code> oder <code>[...] oder [...]</code>
Prüfen	<code>is.list(x)</code>	<code>is.data.frame(x)</code>	<code>tibble::is_tibble(x)</code>
Konvertieren	<code>as.list(x)</code>	<code>as.data.frame(x)</code>	<code>tibble::as_tibble(x)</code>
Dimension <code>dim(x)</code>	1 (NULL)	2 (Zeilen, Spalten)	2 (Zeilen, Spalten)
Länge <code>length(x)</code>	Elemente in der Liste	Anzahl Spalten	Anzahl Spalten
Namen	<code>names(x)</code>	<code>names(x)</code>	<code>names(x)</code>

Objektinformationen

<code>class(x)</code>	Gibt die Klasse von <code>x</code> aus oder setzt sie (<code>class(x) <- "myclass"</code>).
<code>unclass(x)</code>	Entfernt das Klassenattribut von <code>x</code> .
<code>attr(x, which)</code>	Gibt das Attribut <i>which</i> von <code>x</code> aus oder setzt es.
<code>attributes(x)</code>	Gibt die Liste der Attribute von <code>x</code> aus oder setzt sie.
<code>str(x)</code>	Zeigt die interne Struktur eines R-Objekts an.
<code>nrow(x), ncol(x)</code>	Gibt die Anzahl der Zeilen und Spalten von Matrizen, Arrays und Data Frames aus.
<code>ls()</code>	Zeigt die Objekte der aktuellen Umgebung an; mit <i>pattern="pat"</i> kann nach einem Muster gesucht werden.
<code>ls.str()</code>	<code>str()</code> für jedes Objekt der aktuellen Umgebung.
<code>methods(a)</code>	Zeigt die S3-Methoden der generischen Funktion <i>a</i> an.
<code>methods(class=class(a))</code>	Listet alle Methoden auf, die Objekte der Klasse <i>a</i> verarbeiten.
<code>print()</code>	Gibt das Argument in der Konsole aus; generische Funktion.
<code>format(x)</code>	Formatiert ein R-Objekt für die lesbare Ausgabe.

Indizieren und Extrahieren von Daten

Base-R-Ansatz

Index	Erläuterung
Leer	Gibt alle Elemente zurück.
Positiv	Ein Vektor positiver ganzer Zahlen bezeichnet die Positionen der Elemente, die ausgegeben werden sollen.
Negativ	Ein Vektor negativer ganzer Zahlen bezeichnet die Positionen der Elemente, die AUSGELASSEN werden sollen.
Zeichen	Ein Zeichenvektor bezeichnet die Namen der Elemente, die ausgegeben werden sollen.
Logisch	Ein logischer Vektor gibt nur die Elemente zurück, für die der entsprechende Wert TRUE ist.

Vektoren indizieren

Index	Ergebnis
<code>x[2]</code>	2. Element.
<code>x[c(1,4)]</code>	1. und 4. Element.
<code>x[-2]</code>	Alle AUSSER dem 2. Element.
<code>x[-c(1,4)]</code>	Alle AUSSER dem 1. und 4. Element.
<code>x["a"]</code>	Element mit dem Namen <i>a</i> .
<code>x[x>5]</code>	Alle Elemente größer als 5.
<code>x[x %in% c(1,4,13)]</code>	Alle Elemente, die in der angegebenen Menge enthalten sind.

Matrizen indizieren

Index	Ergebnis
<code>x[1:3,]</code>	Zeile 1-3, alle Spalten.
<code>x[, c(2,4)]</code>	Alle Zeilen, Spalte 2 und 4.
<code>x[1,2:4]</code>	Zeile 1, Spalte 2-4.
<code>x["a", c("x", "y")]</code>	Zeile mit dem Namen <i>a</i> , Spalten mit den Namen <i>x</i> oder <i>y</i> .

Listen indizieren

Index	Ergebnis
<code>x[2]</code>	Liste mit dem 2. Objekt.
<code>x[[2]]</code>	2. Objekt.
<code>x[["a"]]</code>	Objekt mit dem Namen <i>a</i> .
<code>x\$a</code>	Objekt mit dem Namen <i>a</i> .

Data Frames/Tibbles indizieren

Wie	bei	Matrizen,	zusätzlich:
Index	Ergebnis		
<code>x[["a"]]</code>	Spalte mit dem Namen <i>a</i> .		
<code>x\$a</code>	Spalte mit dem Namen <i>a</i> .		

Weitere Funktion:

`subset(x, ...)` Gibt Teilmengen von Vektoren, Matrizen oder Data Frames zurück, die die Bedingungen (...) erfüllen.

Tidyverse-Ansatz für Data Frames & Tibbles



`library(dplyr)` oder `library(tidyverse)`

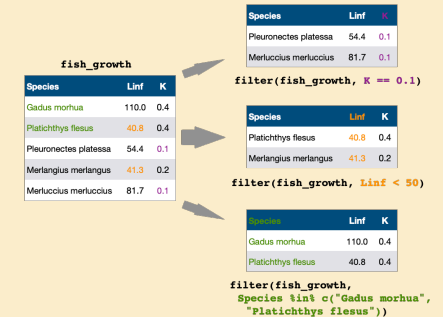
Zeilen extrahieren

`filter(.data, a < 5)`

Extrahiert Zeilen, die logische Kriterien erfüllen (z. B. Werte in Spalte *a* kleiner als 5).

`slice(.data, 5:9)`

Extrahiert Zeilen nach Position (z. B. Zeile 5-9).



Spalten extrahieren

`select(.data, a, b)`

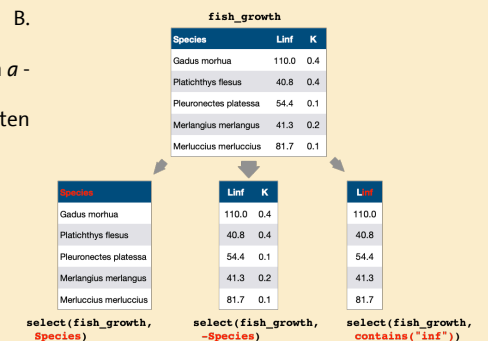
Extrahiert Spalten (z. B. Spalte *a* und *b*).

`select(.data, a:d)`

Extrahiert die Spalten *a* - *d*.

`select(.data, -a)`

Extrahiert alle Spalten AUSSER *a*.



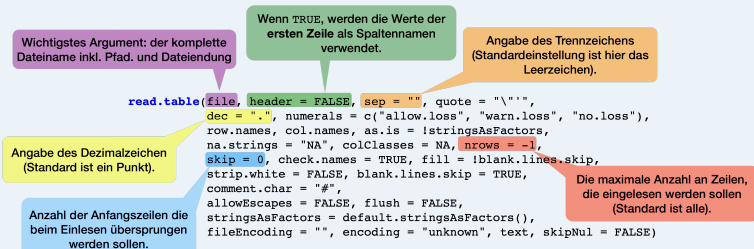
Einlesen und Ausgeben von Daten

Base-R-Funktionen

Einlesen

<code>data(x)</code>	Lädt die angegebenen R-Datensätze.
<code>load(file)</code>	Lädt Objekte, die mit <code>save()</code> geschrieben wurden (Dateiendung: <code>.RData</code> oder <code>.rda</code>).
<code>readRDS(file)</code>	Liest ein einzelnes R-Objekt, das mit <code>saveRDS()</code> geschrieben wurde.
<code>read.table(file, sep=" ", dec=".")</code>	Liest eine Datei im Tabellenformat (durch Leerzeichen getrennt) in einen Data Frame ein.
<code>read.csv(file, sep=";", dec=".")</code>	Liest eine CSV-Datei (kommagetrennt) in einen Data Frame ein.
<code>read.csv2(file, sep=";", dec=",")</code>	Liest eine CSV-Datei (semikolongetrennt) in einen Data Frame ein.
<code>read.delim(file, sep="\t", dec=".")</code>	Liest tabulatorgetrennte Dateien in einen Data Frame ein.
<code>read.delim2(file, sep="\t", dec=",")</code>	Liest tabulatorgetrennte Dateien ein (mit Komma als Dezimalzeichen).
<code>read.fwf(file, widths)</code>	Liest Dateien mit fester Spaltenbreite in einen Data Frame ein; <code>widths</code> ist ein Integer-Vektor mit den Breiten der einzelnen Felder.

Alle Einlesefunktionen haben eine ähnliche Syntax:



Ausgeben

<code>save(x,y, file)</code>	Schreibt ein oder mehrere R-Objekte (z. B. <code>x,y</code>) in die angegebene Datei (Endung <code>.RData</code> oder <code>.rda</code> verwenden). Wie <code>save()</code> , speichert aber alle Objekte des aktuellen Workspace.
<code>save.image(file)</code>	Schreibt ein einzelnes R-Objekt in eine Datei (Einlesen mit <code>readRDS()</code> ; dabei kann ein neuer Name vergeben werden).
<code>saveRDS(x, file)</code>	Schreibt einen Data Frame oder eine Matrix (Tibbles funktionieren ebenfalls) in eine Textdatei; das Format wird durch Argumente wie <code>sep</code> , <code>dec</code> und <code>row.names</code> festgelegt, nicht durch die Dateiendung.
<code>write.table(x,file,sep=" ")</code>	Schreibt einen Data Frame oder eine Matrix in eine CSV-Datei (kommagetrennt); mit <code>row.names = FALSE</code> werden die Zeilenamen weggelassen.
<code>write.csv(x,file)</code>	Schreibt einen Data Frame oder eine Matrix in eine CSV-Datei (semikolongetrennt).
<code>write.csv2(x,file)</code>	Gibt die Objekte nach Umwandlung in Zeichenketten aus; nützlich für die Ausgabe in selbst geschriebenen Funktionen.
<code>cat(..., file, sep=" ")</code>	Leitet die R-Ausgabe in eine Datei um, bis <code>sink()</code> erneut (ohne Argument) aufgerufen wird.
<code>sink(file)</code>	

Zusatzpakete zum Einlesen weiterer Dateitypen

- **readODS** - Calc-Dateien (.ods)
- **readxl** - Excel-Dateien (.xls und .xlsx)
- **haven** - SPSS-, Stata- und SAS-Dateien
- **xml2** - XML
- **jsonlite** - json
- **DBI** - Datenbanken
- **httr2** - Web-APIs
- **rvest** - HTML (Web Scraping)

Tidyverse-Ansatz: das Paket readr



`library(readr)` oder `library(tidyverse)`

Einlesen

Tabellarische oder flache Dateien in Tibbles einlesen:

<code>read_table(file)</code>	Durch Leerraum getrennte Dateien.
<code>read_csv(file)</code>	Kommagetrennte CSV-Dateien.
<code>read_csv2(file)</code>	Semikolongetrennte CSV-Dateien.
<code>read_tsv(file)</code>	Tabulatorgetrennte Dateien.
<code>read_delim(file, delim=" ")</code>	Dateien mit beliebigem Trennzeichen.
<code>read_fwf(file)</code>	Dateien mit fester Spaltenbreite.

Nicht-tabellarische Dateien einlesen:

<code>read_file(file)</code>	Liest die Datei als einzelne Zeichenkette ein.
<code>read_lines(file)</code>	Liest jede Zeile als eigene Zeichenkette ein.
<code>read_log(file)</code>	Liest Logdateien im Apache-Format ein.

Ausgeben

<code>write_csv(x, file)</code>	Schreibt in eine CSV-Datei.
<code>write_excel_csv(x, file)</code>	Schreibt in eine CSV-Datei für Excel.
<code>write_tsv(x, file)</code>	Schreibt in eine tabulatorgetrennte Datei.
<code>write_delim(x, file)</code>	Schreibt in eine Datei mit beliebigem Trennzeichen.
<code>write_file(x, file)</code>	Schreibt eine Zeichenkette in eine Datei.
<code>write_lines(x, file)</code>	Schreibt einen Zeichenvektor in eine Datei, ein Element pro Zeile.
<code>write_rds(x, file)</code>	Schreibt in eine RDS-Datei.

Datenmanipulation

Base-R-Funktionen

Die Base-Funktionen sind überwiegend auf Vektoren ausgelegt, einige lassen sich aber auch auf andere Objekte anwenden:

Elemente anordnen

<code>rev(x)</code>	Kehrt die Reihenfolge der Elemente von x um.
<code>sort(x)</code>	Sortiert die Elemente von x aufsteigend; mit <code>decreasing = TRUE</code> absteigend.
<code>order(x)</code>	Gibt die Indizes zurück, die x aufsteigend sortieren (mit <code>decreasing = TRUE</code> absteigend), z. B. <code>x[order(x)]</code> .

Umgang mit NA

<code>na.omit(x)</code>	Entfernt Beobachtungen mit fehlenden Werten (NA); ist x eine Matrix oder ein Data Frame, wird die gesamte Zeile entfernt.
<code>na.fail(x)</code>	Gibt eine Fehlermeldung aus, falls x mindestens ein NA enthält.

Positionen bestimmen

<code>which(x)</code>	Gibt einen Vektor mit den Indizes aller TRUE-Elemente des logischen Vektors x zurück; x kann ein Vergleich sein (z. B. <code>which(1:5 > 2)</code>).
<code>which.max(x)</code>	Gibt den Index des größten Elements von x zurück.
<code>which.min(x)</code>	Gibt den Index des kleinsten Elements von x zurück.
<code>match(x, y)</code>	Gibt einen Vektor derselben Länge wie x mit den Positionen der (ersten) Übereinstimmungen von x in y zurück (sonst NA).

Objekte verändern

<code>unique(x)</code>	Gibt einen Vektor, Data Frame oder Array wie x zurück, aus dem doppelte Elemente/Zeilen entfernt wurden.
<code>sample(x, size)</code>	Zieht eine Stichprobe der angegebenen Größe <code>size</code> aus den Elementen von x , mit oder ohne Zurücklegen (Argument <code>replace</code>).
<code>expand.grid()</code>	Erzeugt einen Data Frame aus allen Kombinationen der übergebenen Vektoren oder Faktoren.
<code>split(x, f)</code>	Teilt die Daten im Vektor x in die durch f definierten Gruppen auf.
<code>unsplit(value, f)</code>	Macht <code>split()</code> rückgängig und ordnet Elemente oder Zeilen einer Liste von Vektoren oder Data Frames wieder an den durch f vorgegebenen Positionen ein.

Zusammenfassen & Aggregieren

<code>table(x)</code>	Gibt eine Kontingenztafel der Häufigkeiten für jede Kombination von Faktorstufen oder ganzzahligen Werten zurück.
<code>prop.table(x, margin)</code>	Gibt die Einträge einer Tabelle (Vektor, Matrix oder Array) als Anteile der Randsummen zurück; bei einem Vektor wird <code>x/sum(x)</code> berechnet, bei einer Matrix kann <code>margin</code> angegeben werden (1 = Zeilensummen, 2 = Spaltensummen).
<code>aggregate(x, by, FUN)</code>	Teilt die Daten in Teilmengen auf, berechnet für jede eine Kennzahl (angegeben in <code>FUN</code>) und gibt das Ergebnis in übersichtlicher Form zurück.

Tidyverse-Funktionen



Am einfachsten: `library(tidyverse)` lädt alle Kernpakete, darunter `tibble`, `tidyr` und `dplyr`.

Tabellen sortieren

`arrange(.data, ...)` Ordnet die Zeilen nach den angegebenen Variablen.
`desc(x)` Wandelt einen Vektor so um, dass er absteigend sortiert wird; nützlich innerhalb von `arrange()`.

Species	Linf	K
Gadus morhua	110.0	0.4
Platichthys flesus	40.8	0.4
Pleuronectes platessa	54.4	0.1
Merlangius merlangus	41.3	0.2
Merluccius merluccius	81.7	0.1

Species	Linf	K
Platichthys flesus	40.8	0.4
Merlangius merlangus	41.3	0.2
Pleuronectes platessa	54.4	0.1
Merluccius merluccius	81.7	0.1
Gadus morhua	110.0	0.4

Species	Linf	K
Platichthys flesus	40.8	0.4
Gadus morhua	110.0	0.4
Pleuronectes platessa	54.4	0.1
Merlangius merlangus	41.3	0.2
Pleuronectes platessa	54.4	0.1
Merluccius merluccius	81.7	0.1

Umgang mit NA

`drop_na(data, ...)` Entfernt ganze Zeilen mit fehlenden Werten.
`fill(data, ..., .direction)` Füllt fehlende Werte mit dem vorherigen (`.direction = "down"`, Standard) oder dem folgenden (`.direction = "up"`) Eintrag auf. Nützlich, wenn nur Wertänderungen erfasst wurden.
`replace_na(data, replace = list(...))` Ersetzt fehlende Werte durch einen für jede Spalte angegebenen Wert.

station	press	temp
1608	1	14.1
1608	2	NA
1608	3	13.8
1608	4	NA
1608	5	13.6

station	press	temp
1608	1	14.1
1608	2	14.1
1608	3	13.8
1608	4	13.8
1608	5	13.6

station	press	temp
1608	1	14.1
1608	2	-999
1608	3	13.8
1608	4	-999
1608	5	13.6

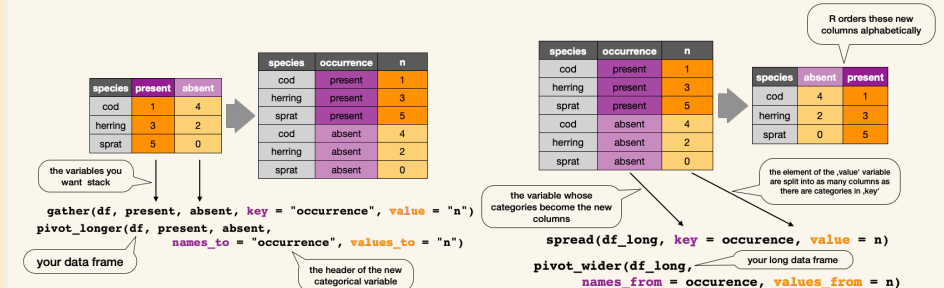
station	press	temp
1608	1	14.1
1608	2	-999
1608	3	13.8
1608	4	-999
1608	5	13.6

Tabellen verändern

Zwischen langem und breitem Format wechseln

`pivot_longer(data, cols, names_to, values_to)`
`pivot_wider(data, names_from, values_from)`

Überführt Daten vom breiten ins lange Format → ersetzt die ältere Funktion `gather()`.
 Überführt Daten vom langen ins breite Format → ersetzt die ältere Funktion `spread()`.



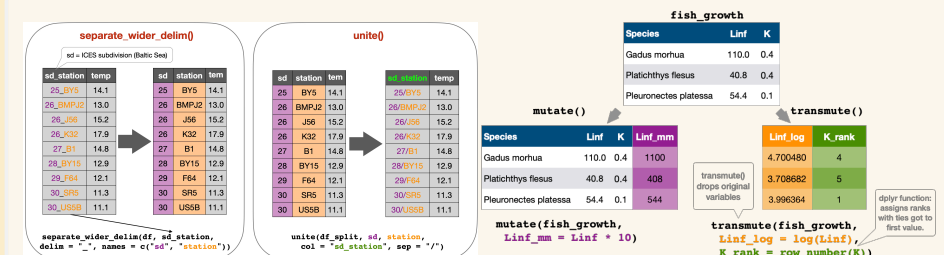
Spalten und Zeilen verändern

`separate_wider_delim(data, cols, delim, names)`
`separate_longer_delim(data, cols, delim)`
`unite(data, col, ..., sep)`

`add_column(data, ...)`
`add_row(data, ...)`
`rename(.data, newname=oldname)`
`mutate(.data, ...)`

`mutate(.data, ..., .keep = "none")`

Teilt eine Zeichenspalte an einem Trennzeichen in mehrere Spalten auf (ersetzt `separate()`).
 Teilt eine Zeichenspalte an einem Trennzeichen in mehrere Zeilen auf (ersetzt `separate_rows()`).
 Fügt mehrere Spalten zu einer zusammen, indem die Zeichenketten aneinandergehängt werden.
 Fügt neue Spalte(n) hinzu (Tibble).
 Fügt eine oder mehrere Zeilen zu einer Tabelle hinzu.
 Benennt Spalten um.
 Fügt neue Variablen hinzu und behält die bestehenden (dplyr).
 Fügt neue Variablen hinzu und verwirft alle bestehenden (ersetzt `transmute()`).



Tabellen erweitern und kombinieren

Tabellen erweitern

`expand_grid()` Erzeugt ein Tibble aus allen Kombinationen der übergebenen Vektoren, Matrizen oder Data Frames (ähnlich der Base-Funktion `expand.grid()`).

`complete(data, ..., fill=list())` Ergänzt einen Data Frame um fehlende Datenkombinationen (alle Variablen bleiben erhalten).

`expand(data, ...)` Erweitert einen Data Frame um alle möglichen Kombinationen der Werte der angegebenen Variablen (alle anderen Variablen werden entfernt).

Zeilen kombinieren

`bind_rows(...)` Fügt mehrere Tabellen unverändert untereinander an (zeilenweise).

`intersect(x, y, ...)` Gibt Zeilen zurück, die sowohl in x UND y vorkommen.

`union(x, y, ...)` Gibt Zeilen zurück, die in x ODER y vorkommen (Duplikate werden entfernt).

`setdiff(x, y, ...)` Gibt Zeilen zurück, die in x, aber NICHT in y vorkommen.

Spalten kombinieren

`bind_cols(...)` Fügt mehrere Tabellen unverändert nebeneinander an (spaltenweise).

Tabellen über gemeinsame Spalten verknüpfen (Joins)

Die Schlüsselspalten werden mit `by = join_by(a)` angegeben (bzw. `join_by(a == b)`, falls die Namen abweichen; `dplyr ≥ 1.1.0`). Unerwartete Many-to-many-Treffer lösen eine Warnung aus; diese lässt sich über das Argument `relationship` steuern.

`left_join(x, y, by = join_by(a))` Fügt Tabelle y anhand gemeinsamer Spaltenwerte an Tabelle x an. Zeilen aus x ohne Treffer in y erhalten NA in den neuen Spalten. Gibt es mehrere Treffer zwischen x und y, werden alle Kombinationen zurückgegeben.

`right_join(x, y, by = join_by(a))` Wie `left_join()`, fügt aber Tabelle x an Tabelle y an.

`full_join(x, y, by = join_by(a))` Verknüpft Tabelle x und y und behält alle Zeilen und Spalten beider Tabellen. Stimmen die Werte der angegebenen Variable nicht überein, werden NAs in den neuen Spalten ergänzt.

`inner_join(x, y, by = join_by(a))` Verknüpft Tabelle x und y, behält aber nur Zeilen, deren Werte in beiden Tabellen übereinstimmen.

Listenspalten erzeugen und auflösen

`nest(.data, ...)` Erzeugt eine Listenspalte aus Data Frames (Nesting).

`unnest(data, cols, ...)` Löst die verschachtelte Listenspalte wieder in reguläre Spalten auf.

Zusammenfassen & Aggregieren

`group_by(.data, ...)` Gruppiert nach einer oder mehreren Variablen; wandelt ein bestehendes Tibble in ein gruppiertes Tibble um, in dem Operationen „nach Gruppen“ ausgeführt werden.

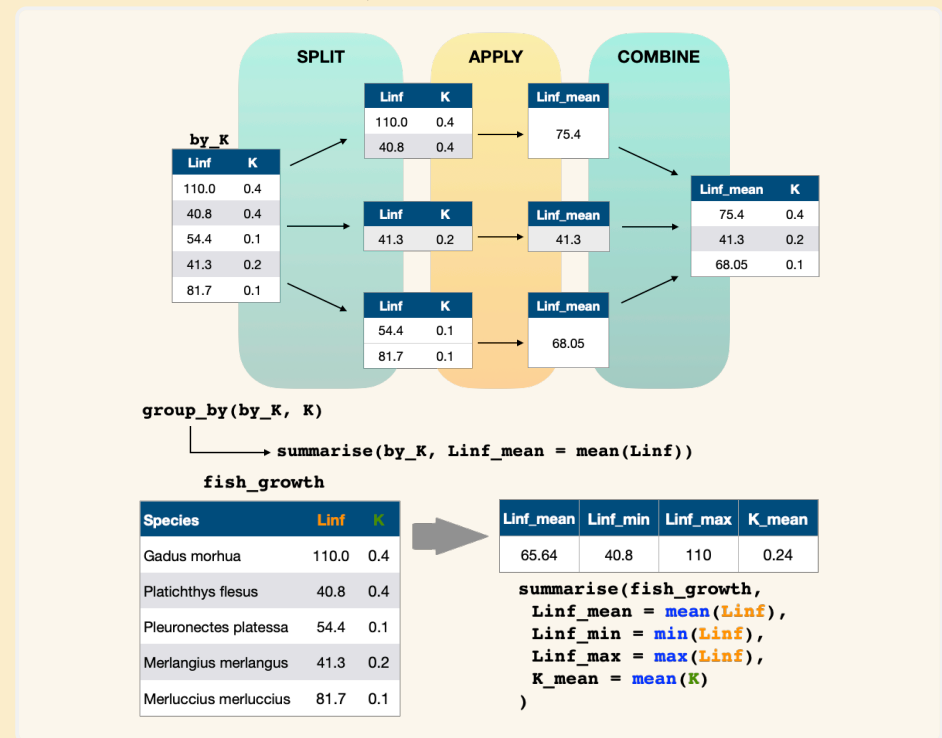
`ungroup(x, ...)` Hebt die Gruppierung auf.

`.by = a` Gruppierung pro Operation als Alternative zu `group_by()` (`dplyr ≥ 1.1.0`), z. B. `summarise(.data, ..., .by = a)`; auch in `mutate()`, `filter()` und `slice()`. Das Ergebnis ist nicht gruppiert.

`summarise(.data, ...)` Erzeugt eine oder mehrere Kennzahlen, die die Variablen eines bestehenden Tibbles mithilfe der angegebenen Funktionen zusammenfassen (z. B. `mean`, `min`, `max`).

`count(x, ...)` Zählt die Beobachtungen in jeder Gruppe einer oder mehrerer Variablen. Kurzform für `group_by(...) |> summarise(n = n())`.

`distinct(.data, ...)` Wählt nur eindeutige Zeilen eines Tibbles aus (ähnlich der Base-Funktion `unique()`), aber schneller und auf bestimmte Spalten beschränkbar.



Nützliche Funktionen für summarise

Die folgenden Base- und dplyr-Funktionen liefern jeweils einen einzelnen Wert (`quantile()` nur bei einem einzelnen Wert in *probs*) und können mit der Base-Funktion `aggregate()` oder der dplyr-Funktion `summarise()` verwendet werden.

Lage:	<code>mean()</code> , <code>median()</code>	Position:	<code>first()</code> , <code>last()</code> , <code>nth()</code> (alle dplyr)
Streuung:	<code>var()</code> , <code>sd()</code> , <code>IQR()</code> , <code>mad()</code>	Anzahl:	<code>n()</code> , <code>n_distinct()</code> (alle dplyr)
Bereich:	<code>min()</code> , <code>max()</code> , <code>quantile()</code>	Logik:	<code>any()</code> , <code>all()</code>