

Umgang mit speziellen Datentypen in R

Faktoren

R verwendet Faktoren zur Darstellung **kategorialer Daten**. Faktoren sind **Integer-Vektoren mit Labels**, die diesen eindeutigen ganzen Zahlen zugeordnet sind. Sie können nur eine **vordefinierte Menge von Werten** enthalten, die sogenannten Levels. Standardmäßig sortiert R die Levels immer alphabetisch. Geordnete Faktoren unterscheiden sich von einfachen Faktoren nur in ihrer Klasse, Methoden und Modellfunktionen behandeln beide Klassen jedoch sehr unterschiedlich.

Base-R-Funktionen

<code>factor(x, levels, labels)</code>	Kodiert einen numerischen, Zeichen- oder logischen Vektor als Faktor; Labels und Levels können optional angegeben werden (über die Angabe der Levels lässt sich die Reihenfolge festlegen).
<code>ordered(x, ..)</code>	Kodiert einen Vektor als geordneten Faktor.
<code>as.factor(x)</code>	Kodiert einen Vektor ebenfalls als Faktor, aber ohne Angabe von Levels und Labels (Kurzform von <code>factor()</code>).
<code>is.factor(x)</code>	Prüft, ob x ein Faktor ist.
<code>as.ordered(x)</code>	Gibt x zurück, falls es geordnet ist, andernfalls <code>ordered(x)</code> .
<code>is.ordered(x)</code>	Prüft, ob x ein geordneter Faktor ist.
<code>addNA(x)</code>	Verändert einen Faktor, indem NA zu einem zusätzlichen Level wird (sodass NA-Werte in Tabellen mitgezählt werden).
<code>levels(x)</code>	Gibt die Levels eines Faktors aus bzw. setzt sie.
<code>droplevels(x)</code>	Entfernt ungenutzte Levels aus Faktoren.
<code>cut(x, breaks)</code>	Wandelt numerische Werte in einen Faktor um, indem der Wertebereich von x in Intervalle unterteilt wird und die Werte entsprechend dem Intervall kodiert werden, in das sie fallen; <code>breaks</code> ist die Anzahl der Intervalle oder ein Vektor mit Schnittpunkten.

Tidyverse-Funktionen



`library(forcats)` oder `library(tidyverse)`

Dieses Paket bietet viele weitere Funktionen zum Verändern und Verarbeiten von Faktoren:

Faktoren untersuchen

<code>fct_count(f)</code>	Zählt die Werte pro Level.
<code>fct_unique(f)</code>	Gibt die eindeutigen Werte zurück und entfernt Duplikate (ähnlich wie <code>unique(f)</code> , aber in der Reihenfolge der Levels, nicht des Auftretens).

Faktoren kombinieren

<code>fct_c(...)</code>	Verkettet Faktoren und kombiniert die Levels.
<code>fct_unify(fs)</code>	Vereinheitlicht die Levels einer Liste von Faktoren (<code>fs</code>).

Reihenfolge der Levels ändern

<code>fct_relevel(.f, ...)</code>	Ordnet Levels manuell neu.
<code>fct_infreq(f)</code>	Ordnet Levels nach der Anzahl der Beobachtungen je Level.
<code>fct_inorder(f)</code>	Ordnet Levels nach der Reihenfolge ihres ersten Auftretens.
<code>fct_inseq(f)</code>	Ordnet Levels nach ihrem numerischen Wert.
<code>fct_rev(f)</code>	Kehrt die Reihenfolge der Levels um.
<code>fct_shift(f)</code>	Verschiebt Levels nach links oder rechts, am Ende wird umgebrochen.
<code>fct_reorder(.f, .x, .fun = median)</code>	Ordnet Levels nach einer anderen Variablen.
<code>fct_reorder2(.f, .x, .y, .fun = last2)</code>	Ordnet Levels nach ihren letzten Werten, wenn sie mit zwei weiteren Variablen dargestellt werden.

Werte von Levels ändern

<code>fct_recode(.f, ...)</code>	Ändert Levels manuell.
<code>fct_relabel(.f, .fun, ...)</code>	Ändert Levels automatisch (folgt der Syntax von <code>purrr::map()</code> , um eine Funktion oder einen Ausdruck auf jedes Level anzuwenden).
<code>fct_collapse(.f, ...)</code>	Fasst Levels zu manuell definierten Gruppen zusammen.
<code>fct_lump_min(f, min)</code>	Fasst Levels zusammen, die weniger als <code>min</code> -mal vorkommen.
<code>fct_lump_prop(f, prop)</code>	Fasst Levels zusammen, die weniger als <code>prop * n</code> -mal vorkommen.
<code>fct_lump_n(f, n)</code>	Fasst alle Levels außer den <code>n</code> häufigsten zusammen.
<code>fct_lump_lowfreq(f)</code>	Fasst die seltensten Levels zusammen, sodass „other“ weiterhin das kleinste Level bleibt.
<code>fct_other(f)</code>	Ersetzt Levels durch „other“.

Levels hinzufügen oder entfernen

<code>fct_drop(f)</code>	Entfernt ungenutzte Levels (anders als <code>base::droplevels()</code> bleiben NA-Levels mit Werten erhalten).
<code>fct_expand(f, ...)</code>	Fügt einem Faktor zusätzliche Levels hinzu.

Datum und Zeit

Datumsangaben werden als Anzahl der Tage seit dem **01.01.1970** dargestellt, frühere Daten haben negative Werte. Sie werden stets nach den Regeln des gregorianischen Kalenders ausgegeben: JJJJ-MM-TT

R verfügt über 3 eingebaute Klassen für Datum und Zeit, auf die Operationen angewendet werden können (siehe ?Ops.Date):

- **Date** → stellt nur Kalenderdaten dar
- **POSIXct** und **POSIXlt** → stellen Kalenderdaten UND Zeit auf die Sekunde genau dar, einschließlich Zeitzone:
 - *POSIXct* stellt intern die (vorzeichenbehaftete) Anzahl der Sekunden seit Beginn des Jahres 1970 (in der Zeitzone UTC) als numerischen Vektor dar (siehe `unclass(as.POSIXct("2020-06-08"))`).
 - *POSIXlt* speichert intern eine benannte Liste von Vektoren (siehe `unclass(as.POSIXlt("2020-06-08"))`). Weitere Informationen unter ?DateTimeClasses.

Base-R-Funktionen

<code>Sys.Date()</code>	Gibt das aktuelle Datum in der aktuellen (System-)Zeitzone zurück (als Klasse <i>Date</i>).
<code>Sys.time()</code>	Gibt einen absoluten Datum-Zeit-Wert zurück, der in verschiedene Zeitzone umgerechnet werden kann (als Klasse <i>POSIXct</i>).
<code>Sys.timezone()</code>	Gibt den Namen der aktuellen Zeitzone zurück.
<code>OlsonNames()</code>	Gibt einen Vektor gültiger Zeitzonennamen zurück.

In Datum oder Datum-Zeit umwandeln

<code>as.Date(x, format="%Y-%m-%d")</code>	Wandelt einen Zeichenvektor in einen <i>Date</i> -Vektor um. Das Argument <i>format</i> legt das Format der Eingabe mithilfe von Prozentzeichen mit Buchstaben fest (Y =Jahr vierstellig, m =Monat zweistellig, d =Tag zweistellig; hier: „2020-05-31“).
<code>as.POSIXct(x)</code>	Wandelt einen Zeichenvektor oder einen Vektor der Klasse <i>Date</i> in die Klasse <i>POSIXct</i> um.
<code>as.POSIXlt(x)</code>	Wandelt in die Klasse <i>POSIXlt</i> um.
<code>strptime(x, format)</code>	Wandelt einen Zeichenvektor mit Datums- und Zeitangaben in die Klasse <i>POSIXlt</i> um.
<code>ISOdate(year, month, day)</code>	Wandelt numerische Werte in die Klasse <i>POSIXct</i> um; als Standardzeit wird 12:00:00 GMT verwendet.
<code>ISOdatetime(year, month, day, hour, min, sec)</code>	Wandelt numerische Werte in die Klasse <i>POSIXct</i> um.
<code>format(x, format)</code>	Wandelt <i>Date</i> - oder <i>POSIXt</i> -Objekte in Zeichen um. Mit dem Argument <i>format</i> lässt sich festlegen, wie Datum und Zeit ausgegeben werden (z. B. <code>format(Sys.Date(), "%d %b %Y")</code>).

Bestandteile extrahieren

<code>quarters(x)</code>	Extrahiert das Quartal eines Jahres aus einem <i>POSIXt</i> - oder <i>Date</i> -Objekt.
<code>months(x)</code>	Extrahiert die Monate eines <i>POSIXt</i> - oder <i>Date</i> -Objekts.
<code>weekdays(x)</code>	Extrahiert die Wochentage.
<code>julian(x, origin)</code>	Extrahiert die julianischen Tage (Tage seit einem Ursprungsdatum).

Tidyverse-Funktionen



`library(lubridate)` oder `library(tidyverse)`

Das Paket *lubridate* stellt Funktionen bereit, die den Umgang mit Datum und Zeit deutlich vereinfachen im Vergleich zu den Base-Funktionen:

<code>today()</code>	Gibt das aktuelle Datum in der aktuellen (System-)Zeitzone zurück (als Klasse <i>Date</i>).
<code>now()</code>	Gibt aktuelles Datum und Uhrzeit als <i>POSIXct</i> -Objekt zurück.

In Datum oder Datum-Zeit umwandeln

<code>as_date(x, ..)</code>	Wandelt einen Vektor aus <i>POSIXt</i> -, numerischen oder Zeichenobjekten in Datumsangaben um (Alternative zu <code>base::as.Date()</code>).
<code>as_datetime(x, ..)</code>	Wandelt einen Vektor aus <i>Date</i> -, numerischen oder Zeichenobjekten in Datum-Zeit-Werte um (Alternative zu <code>base::as.POSIXct()</code>).
<code>make_date(year, month, day)</code>	Wandelt numerische Werte in Objekte der Klasse <i>Date</i> um.
<code>make_datetime(year, month, day, hour, min, sec)</code>	Wandelt numerische Werte in Objekte der Klasse <i>POSIXct</i> um (schnellere Version von <code>base::ISOdatetime()</code> ; Standardzeitzone ist UTC).
<code>ymd(x), ydm(x), mdy(x), myd(x), dmy(x), dym(x)</code>	Liegt das Datum im Zeichenvektor nicht im Standardformat vor (z. B. anderes Trennzeichen oder andere Reihenfolge), sind diese Funktionen zum Einlesen besser geeignet. Die Reihenfolge der 3 Bestandteile Jahr (y), Monat (m) und Tag (d) bestimmt, welche Funktion zu verwenden ist. Beispiel: <code>myd("June, 2020, 16")</code>
<code>XXX_hms(x), XXX_hm(x), XXX_h(x)</code>	Jede der obigen 6 Einlesefunktionen, kombiniert mit <code>_hms</code> , <code>_hm</code> , <code>_h</code> , liest Datum-Zeit-Werte mit Jahr, Monat und Tag sowie Stunde (h), Minute (m) und Sekunde (s) ein. Beispiel: <code>dmy_hm("31/05/2020 1:02")</code>

Bestandteile auslesen und setzen

Die folgenden Funktionen extrahieren (oder ändern) Bestandteile von Datums- und Datum-Zeit-Objekten:

`year(x)`, `semester(x)`, `quarter(x)`, `month(x)`, `week(x)`, `day(x)`, `wday(x)`, `yday(x)`, `hour(x)`, `minute(x)`, `second(x)`, `tz(x)` (=Zeitzone)

Beispiele:

<code>x <- as_datetime("2020-05-28 14:00:00 UTC")</code>	
<code>wday(x, label=TRUE)</code>	Extrahiert den Wochentag (hier als Name).
<code>tz(x) <- "America/New_York"</code>	Ersetzt die Zeitzone und behält die Uhrzeit bei (Alias für <code>force_tz()</code>); der Zeitpunkt ändert sich.
<code>with_tz(x, "America/New_York")</code>	Ändert die Zeitzone und behält den Zeitpunkt bei; die angezeigte Uhrzeit ändert sich.

Zeitoperationen

<code>interval(start, end)</code> oder <code>start %--% end</code> <code>int_length(int)</code> <code>duration(num, units)</code>	Erzeugt ein <i>Interval</i> -Objekt mit den angegebenen Start- und Enddaten. Beispiel: <code>dmy("10/05/2020") %--% dmy("17/05/2020")</code> Gibt die Länge des Intervalls in Sekunden zurück. Erzeugt ein <i>duration</i> -Objekt (=exakte Zeitmessung); Standard ist Sekunden.
<code>as.duration(x, ..)</code>	Wandelt ein Intervall (oder eine Periode bzw. einen numerischen Wert) in ein <i>duration</i> -Objekt um. Beispiel: <code>as.duration(int)</code> für ein Intervall <i>int</i> ; ähnlich wie <code>int_length(int)</code>
<code>period(num, units)</code>	Erzeugt ein <i>period</i> -Objekt (=Zeitspanne in Uhrzeit-/Kalendereinheiten, z. B. <code>days(1)</code> , <code>months(2)</code>). Im Gegensatz zu Durations können Perioden unterschiedlich lang sein (z. B. wegen Schaltjahren oder Sommerzeit).
<code>make_difftime(num, units)</code>	Erzeugt ein <i>difftime</i> -Objekt mit der angegebenen Anzahl an Einheiten.

Zeichenketten (Strings)

Tidverse-Ansatz zur Bearbeitung von Zeichenketten (vs. base R)



`library(stringr)` oder `library(tidyverse)`

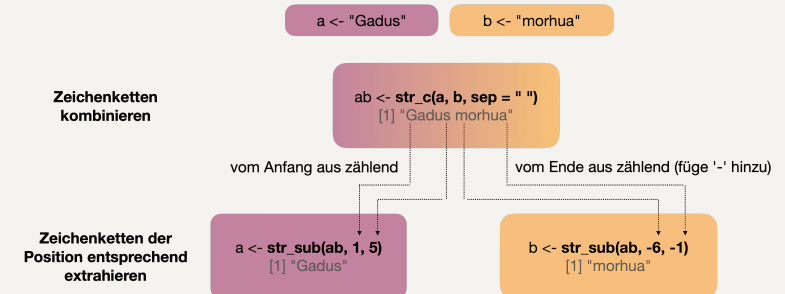
Einige stringr-Funktionen haben eine äquivalente Base-Funktion:

Zeichenketten prüfen & verändern

stringr-Funktionen	Beschreibung	Base-Funktionen
<code>str_length(string)</code>	Anzahl der Zeichen in der Zeichenkette.	<code>nchar(x)</code>
<code>str_to_upper(string)</code>	Wandelt in Großbuchstaben um.	<code>toupper(x)</code>
<code>str_to_lower(string)</code>	Wandelt in Kleinbuchstaben um.	<code>tolower(x)</code>
<code>str_to_title(string)</code>	Wandelt in Titelschreibweise um.	<code>tools::toTitleCase(x)</code>
<code>str_trim(string, side)</code>	Entfernt Leerraum am Anfang und/oder Ende einer Zeichenkette.	<code>trimws(x, which)</code>
<code>str_squish(string)</code>	Entfernt Leerraum am Anfang und Ende und reduziert mehrfachen Leerraum im Inneren auf ein Leerzeichen.	
<code>str_pad(string, width)</code>	Füllt Zeichenketten auf eine konstante Breite auf.	
<code>str_trunc(string, width)</code>	Kürzt Zeichenketten auf die angegebene Breite.	

Zusammenfügen

stringr-Funktionen	Beschreibung	Base-Funktionen
<code>str_c(x,y, sep=' ')</code>	Fügt mehrere Vektoren zusammen (Standard <code>sep = " "</code> ; NA bleibt NA).	<code>paste(x,y, sep=' ')</code>
<code>str_c(x, collapse=' ')</code>	Fügt die Elemente eines Vektors zusammen.	<code>paste(x, collapse=' ')</code>



Zeichenketten sortieren

stringr-Funktionen	Beschreibung	Base-Funktionen
<code>str_sort(x)</code>	Sortiert Zeichenketten alphabetisch.	<code>sort(x)</code>
<code>str_order(x)</code>	Gibt den Vektor der Indizes zurück, der einen Zeichenvektor sortiert.	<code>order(x)</code>

Muster finden & lokalisieren

stringr-Funktionen	Beschreibung	Base-Funktionen
<code>str_detect(string, pattern)</code>	Findet Muster in der Zeichenkette, gibt einen logischen Vektor zurück.	<code>grepl(pattern, x)</code>
<code>str_which(string, pattern)</code>	Gibt die Position der Zeichenketten zurück, die das Muster enthalten (Wrapper für <code>which(str_detect(string, pattern))</code>).	<code>grep(pattern, x)</code>
<code>str_count(string, pattern)</code>	Zählt die Treffer in einer Zeichenkette.	
<code>str_locate(string, pattern)</code>	Gibt Start- und Endposition des ersten Treffers in jeder Zeichenkette aus, das Ergebnis ist eine Matrix.	<code>regexpr(pattern, x)</code>
<code>str_locate_all(string, pattern)</code>	Gibt Start- und Endposition aller Treffer in jeder Zeichenkette aus, das Ergebnis ist eine Liste.	<code>gregexpr(pattern, x)</code>
<code>str_view(string, pattern)</code>	Zeigt die Treffer eines Musters in jeder Zeichenkette im Viewer an (nützlich zum Testen regulärer Ausdrücke).	

Muster extrahieren & Teilmengen bilden

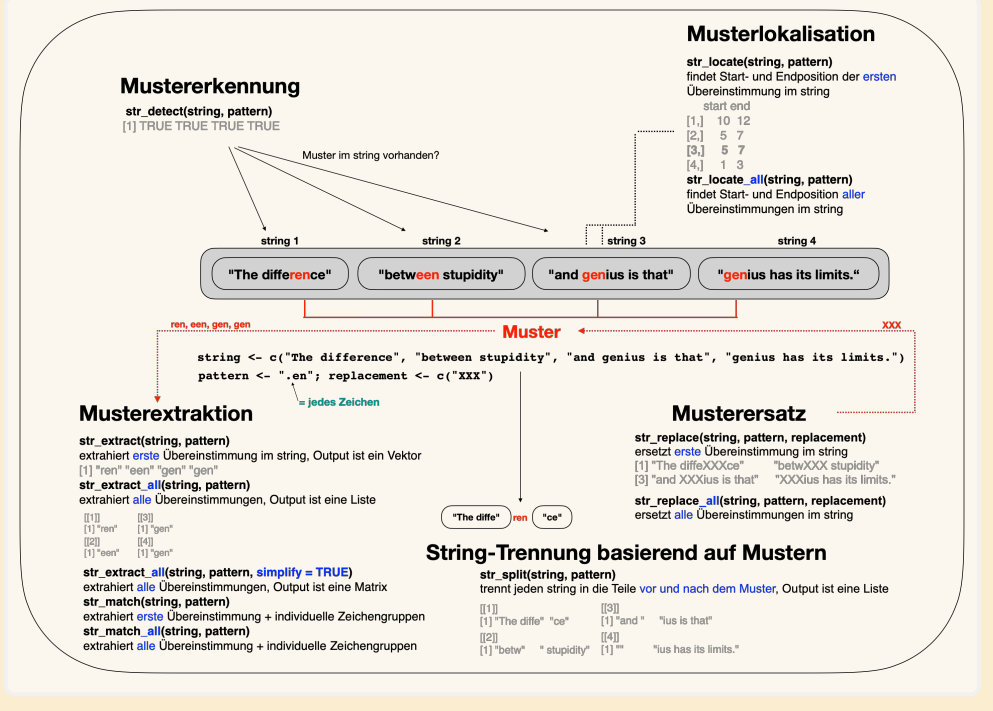
stringr-Funktionen	Beschreibung	Base-Funktionen
<code>str_sub(string, start, end)</code>	Extrahiert und ersetzt Teilzeichenketten an einer bestimmten Position in einem Zeichenvektor.	<code>substr(x, start, stop)</code>
<code>str_extract(string, pattern)</code>	Gibt den ersten Treffer in jeder Zeichenkette als Vektor zurück.	
<code>str_extract_all(string, pattern, simplify)</code>	Gibt jeden Treffer zurück, das Ergebnis ist eine Matrix (bei <code>simplify=TRUE</code>) oder Liste.	
<code>str_subset(string, pattern)</code>	Wählt Zeichenketten aus, die ein bestimmtes Muster enthalten (Wrapper um <code>x[str_detect(x, pattern)]</code>).	<code>grep(pattern, x, value=TRUE)</code>
<code>str_match(string, pattern)</code>	Gibt den ersten Treffer in jeder Zeichenkette als Matrix mit einer Spalte je Gruppe im Muster zurück.	
<code>str_match_all(string, pattern)</code>	Gibt alle gefundenen Treffer als Liste von Matrizen zurück.	

Muster ersetzen

stringr-Funktionen	Beschreibung	Base-Funktionen
<code>str_replace(string, pattern, replacement)</code>	Ersetzt den ersten Treffer.	<code>sub(pattern, replacement, x)</code>
<code>str_replace_all(string, pattern, replacement)</code>	Ersetzt alle Treffer.	<code>gsub(pattern, replacement, x)</code>

Anhand von Mustern aufteilen

stringr-Funktionen	Beschreibung	Base-Funktionen
<code>str_split(string, pattern)</code>	Teilt die Elemente eines Zeichenvektors auf, gibt eine Liste zurück.	<code>strsplit(x, split)</code>



Reguläre Ausdrücke

Zeichenklassen	
<code>[:digit:]</code> oder <code>\d</code>	Ziffern; [0-9]
<code>\D</code>	Nicht-Ziffern; [^0-9]
<code>[:lower:]</code>	Kleinbuchstaben; [a-z]
<code>[:upper:]</code>	Großbuchstaben; [A-Z]
<code>[:alpha:]</code>	Buchstaben; [A-Za-z]
<code>[:alnum:]</code>	Alphanumerische Zeichen; [A-Za-z0-9]
<code>\w</code>	Wortzeichen; [A-Za-z0-9_]
<code>\W</code>	Nicht-Wortzeichen
<code>[:xdigit:]</code> oder <code>\x</code>	Hexadezimalziffern; [0-9A-Fa-f]
<code>[:blank:]</code>	Leerzeichen und Tabulator
<code>[:space:]</code> oder <code>\s</code>	Leerzeichen, Tabulator, vertikaler Tabulator, Zeilenumbruch, Seitenvorschub, Wagenrücklauf
<code>\S</code>	Kein Leerzeichen; [^[:space:]]
<code>[:punct:]</code>	Satzzeichen; !"#\$%&'()*+,-./:;<=>?@[^_`{ }~
<code>[:graph:]</code>	Grafische Zeichen; [[:alnum:][:punct:]]
<code>[:print:]</code>	Druckbare Zeichen [[:alnum:][:punct:][:space:]]
<code>[:cntrl:]</code> oder <code>\c</code>	Steuerzeichen \n, \r etc.
Zeichenklassen und Gruppen	
<code>.</code>	Beliebiges Zeichen AUSSER \n
<code> </code>	ODER, z. B. (alb)
<code>[...]</code>	Erlaubte Zeichen auflisten, z.B. [abc]
<code>[a-z]</code>	Zeichenbereiche angeben
<code>[^...]</code>	Ausgeschlossene Zeichen auflisten
<code>(...)</code>	Gruppierung, ermöglicht Rückbezüge mit <code>\N</code> , wobei N eine ganze Zahl ist

Anker (anchors)	
<code>^</code>	Anfang der Zeichenkette
<code>\$</code>	Ende der Zeichenkette
<code>\b</code>	Leere Zeichenkette an einem Wortrand (Anfang oder Ende)
<code>\B</code>	KEIN Wortrand
<code>\<</code>	Wortanfang
<code>\></code>	Wortende

Quantifizierer	
<code>*</code>	Mindestens 0-mal
<code>+</code>	Mindestens 1-mal
<code>?</code>	Höchstens 1-mal; optionale Zeichenkette
<code>{n}</code>	Genau n-mal
<code>{n,}</code>	Mindestens n-mal
<code>{,n}</code>	Höchstens n-mal
<code>{n,m}</code>	Zwischen n- und m-mal

Spezielle Metazeichen	
<code>\n</code>	Zeilenumbruch
<code>\r</code>	Wagenrücklauf
<code>\t</code>	Tabulator
<code>\v</code>	Vertikaler Tabulator
<code>\f</code>	Seitenvorschub

Zeichen maskieren (Escaping):

Metazeichen (`.` `*` `+` usw.) können als wörtliche Zeichen verwendet werden, indem man sie maskiert. Zeichen lassen sich mit `\` maskieren oder indem man sie in `\Q...\E` einschließt.