

Handling of special data types in R

Factors

R uses factors to represent **categorical data**. Factors are **integer vectors with labels** associated with these unique integers. They can only contain a **pre-defined set of values**, known as levels. By default, R always sorts levels in alphabetical order. Ordered factors differ from factors only in their class, but methods and the model-fitting functions treat the two classes quite differently.

Base functions

<code>factor(x, levels, labels)</code>	Encode a numerical, character or logical vector as factor; here, labels and levels can optionally be specified (specifying levels allows you to determine the order).
<code>ordered(x, ..)</code>	Encode a vector as an ordered factor.
<code>as.factor(x)</code>	Also encode a vector as factor but without specifying levels and labels (abbreviated form of <code>factor()</code>).
<code>is.factor(x)</code>	Check whether <code>x</code> is a factor.
<code>as.ordered(x)</code>	Return <code>x</code> if this is ordered, and <code>ordered(x)</code> otherwise.
<code>is.ordered(x)</code>	Check whether <code>x</code> is an ordered factor.
<code>addNA(x)</code>	Modify a factor by turning NA into an extra level (so that NA values are counted in tables).
<code>levels(x)</code>	Return/set the levels of a factor.
<code>droplevels(x)</code>	Drop unused levels from factors.
<code>cut(x, breaks)</code>	Convert numeric to factor by dividing the range of <code>x</code> into intervals and coding the values in <code>x</code> according to which interval they fall into; <code>breaks</code> is the number of cut intervals or a vector of cut points.

Tidyverse functions



`library(forcats)` or `library(tidyverse)`

This package provides many more functions to change and handle factors:

Inspect factors

<code>fct_count(f)</code>	Count the number of values with each level.
<code>fct_unique(f)</code>	Return the unique values, removing duplicates (similar to <code>unique(f)</code> but in order of levels, not appearance).

Combine factors

<code>fct_c(...)</code>	Concatenate factors, combining levels.
<code>fct_unify(fs)</code>	Unify the levels across a list of factors (<code>fs</code>).

Change order of factor levels

<code>fct_relevel(.f, ...)</code>	Manually reorder levels.
<code>fct_infreq(f)</code>	Reorder levels by number of observations within each level.
<code>fct_inorder(f)</code>	Reorder levels by the order in which they first appear.
<code>fct_inseq(f)</code>	Reorder levels by numeric value of level.
<code>fct_rev(f)</code>	Reverse order of levels.
<code>fct_shift(f)</code>	Shift levels to left or right, wrapping around end.
<code>fct_reorder(.f, .x, .fun = median)</code>	Reorder levels by sorting along another variable.
<code>fct_reorder2(.f, .x, .y, .fun = last2)</code>	Reorder levels by their final values when plotted with two other variables.

Change values of levels

<code>fct_recode(.f, ...)</code>	Manually change factor levels.
<code>fct_relabel(.f, .fun, ...)</code>	Automatically change factor levels (obeys <code>purrr::map()</code> syntax to apply a function or expression to each level).
<code>fct_collapse(.f, ...)</code>	Collapse levels into manually defined groups.
<code>fct_lump_min(f, min)</code>	Lump levels that appear fewer than <i>min</i> times.
<code>fct_lump_prop(f, prop)</code>	Lump levels that appear in fewer <i>prop</i> * <i>n</i> times.
<code>fct_lump_n(f, n)</code>	Lump all levels except for the <i>n</i> most frequent.
<code>fct_lump_lowfreq(f)</code>	Lump together the least frequent levels, ensuring that "other" is still the smallest level.
<code>fct_other(f)</code>	Replace levels with "other".

Add or drop levels

<code>fct_drop(f)</code>	Drop unused levels (unlike <code>base::droplevels()</code> , NA levels that have values are kept).
<code>fct_expand(f, ...)</code>	Add additional levels to a factor.

Dates and Times

Dates are represented as the number of days since **1970-01-01**, with negative values for earlier dates. They are always printed following the rules of the current Gregorian calendar: YYYY-MM-DD

R has 3 built-in classes for date and date-time on which operations can be applied (see [?Ops.Date](#)):

- **Date** → represents only calendar dates
- **POSIXct** and **POSIXlt** → represent calendar dates AND time to the nearest second, including time zones:
 - *POSIXct* represents the (signed) number of seconds since the beginning of 1970 (in the UTC time zone) as a numeric vector internally (see `unclass(as.POSIXct("2020-06-08"))`).
 - *POSIXlt* stores internally a named list of vectors (see `unclass(as.POSIXlt("2020-06-08"))`). For more information check [?DateTimeClasses](#).

Base functions

<code>Sys.Date()</code>	Return the current date in the current time zone (of the system) (as class <i>Date</i>).
<code>Sys.time()</code>	Return an absolute date-time value which can be converted to various time zones (as class <i>POSIXct</i>).
<code>Sys.timezone()</code>	Return the name of the current time zone.
<code>OlsonNames()</code>	Return a vector of valid time zone names.

Convert to date or date-time

<code>as.Date(x, format="%Y-%m-%d")</code>	Convert character vector into a <i>Date</i> vector. The <i>format</i> argument specifies the format of the input using percent symbols with characters (Y =year with 4 digits, m =month with 2 digits, d =day with 2 digits; here: "2020-05-31").
<code>as.POSIXct(x)</code>	Convert a character vector or vector of class <i>Date</i> into the <i>POSIXct</i> class.
<code>as.POSIXlt(x)</code>	Convert into the <i>POSIXlt</i> class.
<code>strptime(x, format)</code>	Convert character vector with date and time info into <i>POSIXlt</i> class.
<code>ISOdate(year, month, day)</code>	Convert numerical values into <i>POSIXct</i> class; uses as default time 12:00:00 GMT.
<code>ISOdatetime(year, month, day, hour, min, sec)</code>	Convert numerical values into <i>POSIXct</i> class.
<code>format(x, format)</code>	Convert <i>Date</i> or <i>POSIXct</i> objects into character. The <i>format</i> argument allows to specify how the date and time should be returned (e.g. <code>format(Sys.Date(), "%d %b %Y")</code>).

Get components

<code>quarters(x)</code>	Extract the annual quarter of a <i>POSIXct</i> or <i>Date</i> object.
<code>months(x)</code>	Extract the months of a <i>POSIXct</i> or <i>Date</i> object.
<code>weekdays(x)</code>	Extract the weekdays.
<code>julian(x, origin)</code>	Extract the Julian days (days since some origin).

Tidyverse functions



`library(lubridate)` or `library(tidyverse)`

The *lubridate* package provides functions that make handling of dates and times much easier than the base functions:

<code>today()</code>	Return the current date in the current time zone (of the system) (as class <i>Date</i>).
<code>now()</code>	Return the current date and time as <i>POSIXct</i> object.

Convert to date or date-time

<code>as_date(x, ...)</code>	Convert a vector of <i>POSIXt</i> , numeric or character objects into dates (alternative to <code>base::as.Date()</code>).
<code>as_datetime(x, ...)</code>	Convert a vector of <i>Date</i> , numeric or character objects into date-times (alternative to <code>base::as.POSIXct()</code>).
<code>make_date(year, month, day)</code>	Convert numerical values into objects of class <i>Date</i> .
<code>make_datetime(year, month, day, hour, min, sec)</code>	Convert numerical values into objects of class <i>POSIXct</i> (faster version of <code>base::ISOdatetime()</code> ; default time zone is UTC).
<code>ymd(x)</code> , <code>ydm(x)</code> , <code>mdy(x)</code> , <code>myd(x)</code> , <code>dmy(x)</code> , <code>dym(x)</code>	If date in character vector is not in the standard format (e.g. different separator or order), these functions are more suitable to parse dates. The order of the 3 components year (y), month (m), and day (d) determines the function to be used. Example: <code>myd("June,2020,16")</code>
<code>XXX_hms(x)</code> , <code>XXX_hm(x)</code> , <code>XXX_h(x)</code>	Any of the above 6 parsing functions combined with <code>_hms</code> , <code>_hm</code> , <code>_h</code> parse date-times with year, month, and day, hour (h), minute (m), and second (s) components. Example: <code>dmy_hm("31/05/2020 1:02")</code>

Get and set components

The following functions extract (or change) components from date and date-time objects:

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

Examples:

<code>x <- as_datetime("2020-05-28 14:00:00 UTC")</code>	
<code>wday(x, label=TRUE)</code>	Extracts the day of the week (here by name).
<code>tz(x) <- "Europe/Berlin"</code>	Replaces the time zone and keeps the clock time (alias for <code>force_tz()</code>); the instant in time changes.
<code>with_tz(x, "Europe/Berlin")</code>	Changes the time zone and keeps the instant in time; the displayed clock time changes.

Time operations

<code>interval(start, end)</code> or <code>start %--% end</code>	Create an <i>Interval</i> object with the specified start and end dates. Example: <code>dmy("10/05/2020") %--% dmy("17/05/2020")</code>
<code>int_length(int)</code>	Return the length of the interval in seconds.
<code>duration(num, units)</code>	Create a <i>duration</i> object (=exact time measurements); default is in seconds.
<code>as.duration(x,...)</code>	Coerce an interval (or a period or numeric value) into a <i>duration</i> object. Example: <code>as.duration(int)</code> for an interval <i>int</i> ; similar to <code>int_length(int)</code>
<code>period(num, units)</code>	Create a <i>period</i> object (=time span in clock time, e.g. <code>days(1)</code> , <code>months(2)</code>). In contrast to durations, periods can vary in length (e.g. due to leap years or daylight saving time).
<code>make_difftime(num, units)</code>	Create a <i>difftime</i> object with the specified number of units.

Strings

Tidyverse approach to string manipulation (vs. base R)



`library(stringr)` or `library(tidyverse)`

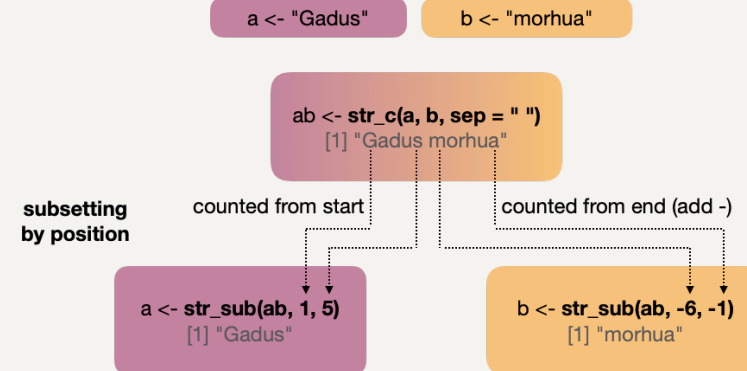
Some stringr functions have an equivalent base function:

Check & modify strings

stringr functions	Description	Base functions
<code>str_length(string)</code>	Number of characters in string.	<code>nchar(x)</code>
<code>str_to_upper(string)</code>	Convert to upper case.	<code>toupper(x)</code>
<code>str_to_lower(string)</code>	Convert to lower case.	<code>tolower(x)</code>
<code>str_to_title(string)</code>	Convert to title case.	<code>tools::toTitleCase(x)</code>
<code>str_trim(string, side)</code>	Trim whitespace from the start and/or end of a string.	<code>trimws(x, which)</code>
<code>str_squish(string)</code>	Trim whitespace from the start and end and reduce repeated inner whitespace to a single space.	
<code>str_pad(string, width)</code>	Pad strings to constant width.	
<code>str_trunc(string, width)</code>	Truncate the width of strings.	

Combine

stringr functions	Description	Base functions
<code>str_c(x,y, sep=' ')</code>	Join multiple vectors together (default <code>sep = " "</code> ; NA stays NA).	<code>paste(x,y, sep=' ')</code>
<code>str_c(x, collapse=' ')</code>	Join vector elements together.	<code>paste(x, collapse=' ')</code>



Order strings

stringr functions	Description	Base functions
<code>str_sort(x)</code>	Sort strings in alphanumerical order.	<code>sort(x)</code>
<code>str_order(x)</code>	Return the vector of indexes that sorts a character vector.	<code>order(x)</code>

Detect & locate patterns

stringr functions	Description	Base functions
<code>str_detect(string, pattern)</code>	Detect patterns in string, returns logical vector.	<code>grepl(pattern, x)</code>
<code>str_which(string, pattern)</code>	Return position of strings that contain a pattern match (wrapper for <code>which(str_detect(string, pattern))</code>).	<code>grep(pattern, x)</code>
<code>str_count(string, pattern)</code>	Count the number of matches in a string.	
<code>str_locate(string, pattern)</code>	Find starting and end position of first match in each string, output is a matrix.	<code>regexpr(pattern, x)</code>
<code>str_locate_all(string, pattern)</code>	Find starting and end position of all matches within each string, output is a list.	<code>gregexpr(pattern, x)</code>
<code>str_view(string, pattern)</code>	Show the matches of a pattern in each string in the viewer (useful for testing regular expressions).	

Extract & subset patterns

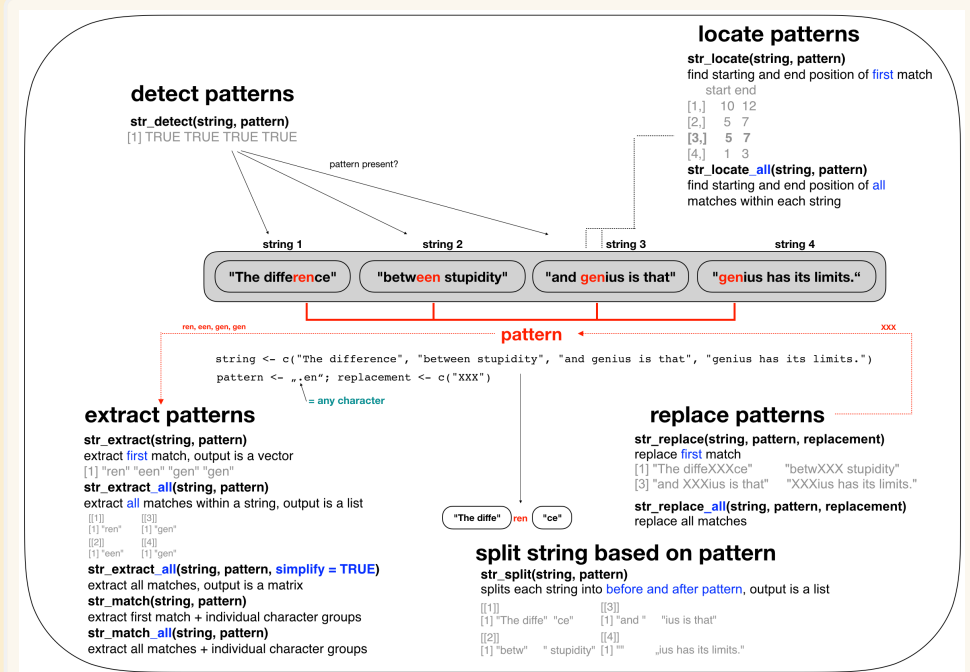
stringr functions	Description	Base functions
<code>str_sub(string, start, end)</code>	Extract and replace substrings at a specific position from a character vector.	<code>substr(x, start, stop)</code>
<code>str_extract(string, pattern)</code>	Return the first pattern match found in each string, as a vector.	
<code>str_extract_all(string, pattern, simplify)</code>	Return every pattern match, output is a matrix (if <code>simplify=TRUE</code>) or list.	
<code>str_subset(string, pattern)</code>	Subset strings that match a specific pattern (wrapper around <code>x[str_detect(x, pattern)]</code>).	<code>grep(pattern, x, value=TRUE)</code>
<code>str_match(string, pattern)</code>	Return the first pattern match found in each string, as a matrix with a column for each group in pattern.	
<code>str_match_all(string, pattern)</code>	Return all pattern matches found, as a list of matrices.	

Replace patterns

stringr functions	Description	Base functions
<code>str_replace(string, pattern, replacement)</code>	Perform replacement of first match.	<code>sub(pattern, replacement, x)</code>
<code>str_replace_all(string, pattern, replacement)</code>	Perform replacement of all matches.	<code>gsub(pattern, replacement, x)</code>

Split based on patterns

stringr functions	Description	Base functions
<code>str_split(string, pattern)</code>	Split elements of a character vector, returns list.	<code>strsplit(x, split)</code>



Regular expressions

Character classes	
<code>[[:digit:]]</code> or <code>\d</code>	Digits; [0-9]
<code>\D</code>	Non-digits; [^0-9]
<code>[[:lower:]]</code>	Lower-case letters; [a-z]
<code>[[:upper:]]</code>	Upper-case letters; [A-Z]
<code>[[:alpha:]]</code>	Alphabetic characters; [A-z]
<code>[[:alnum:]]</code>	Alphanumeric characters [A-z0-9]
<code>\w</code>	Word characters; [A-z0-9_]
<code>\W</code>	Non-word characters
<code>[[:xdigit:]]</code> or <code>\x</code>	Hexadec. digits; [0-9A-Fa-f]
<code>[[:blank:]]</code>	Space and tab
<code>[[:space:]]</code> or <code>\s</code>	Space, tab, vertical tab, newline, form feed, carriage return
<code>\S</code>	Not space; [^[:space:]]
<code>[[:punct:]]</code>	Punctuation characters; !"#\$%&'()*+,-./:;<=>? @[^_`{}~
<code>[[:graph:]]</code>	Graphical char.; [[:alnum:]][[:punct:]]
<code>[[:print:]]</code>	Printable characters; [[:alnum:]][[:punct:]]\s]
<code>[[:cntrl:]]</code> or <code>\c</code>	Control characters; \n, \r etc.
Character classes and groups	
<code>.</code>	Any character except \n
<code> </code>	Or, e.g. (alb)
<code>[...]</code>	List permitted characters, e.g. [abc]
<code>[a-z]</code>	Specify character ranges
<code>[^...]</code>	List excluded characters
<code>(...)</code>	Grouping, enables back referencing using \N where N is an integer

Anchors	
<code>^</code>	Start of the string
<code>\$</code>	End of the string
<code>\b</code>	Empty string at either edge of a word
<code>\B</code>	NOT the edge of a word
<code>\<</code>	Beginning of a word
<code>\></code>	End of a word

Quantifiers	
<code>*</code>	Matches at least 0 times
<code>+</code>	Matches at least 1 time
<code>?</code>	Matches at most 1 time; optional string
<code>{n}</code>	Matches exactly n times
<code>{n,}</code>	Matches at least n times
<code>{,n}</code>	Matches at most n times
<code>{n,m}</code>	Matches between n and m times

Special Metacharacters	
<code>\n</code>	New line
<code>\r</code>	Carriage return
<code>\t</code>	Tab
<code>\v</code>	Vertical tab
<code>\f</code>	Form feed

Escaping characters:

Metacharacters (`.` `*` `+` etc.) can be used as literal characters by escaping them. Characters can be escaped using `\` or by enclosing them in `\Q...\\E`.