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

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

Tidyverse functions

`load(forcats)` or `load(tidyverse)`

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

Inspect factors

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

Combine factors

`fct_c(..)`	Concatenate factors, combining levels.
`fct_unify(fs)`	Unify the levels across a list of factors (*fs*).

Change order of factor levels

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

Change values of levels

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

Add or drop levels

`fct_drop(f)`	Drop unused levels (but not NA levels that have values as `base::droplevels()` does).
`fct_expand(f,..)`	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

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

Convert to date or date-time

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

Get components

`quarters(x)`	Extract the annual quarter of a *POSIXt* or *Date* Object.
`months(x)`	Extract the months of a *POSIXt* or *Date* Object.
`weekdays(x)`	Extract the weekdays.
`julian(x, origin)`	Extract the Julian days (days since some origin).

Tidyverse functions

`load(lubridate)`

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

`today()`	Return the current date in the current time zone (of the system) (as class *Date*).
`now()`	Return the current date and time as *POSIXct* object.

Convert to date or date-time

`as_date(x,..)`	Convert a vector of POSIXt, numeric or character objects into dates (alternative to `base::as.Date()`).
`as_datetime(x,..)`	Convert a vector of Date, numeric or character objects into dates (alternative to `base::as.POSIXct()`).
`make_date(year, month,day,hour)`	Convert numerical values into objects of class *Date*.
`make_datetime(year, month,day,hour,min, sec)`	Convert numerical values into objects of class *POSIXct* (faster version of `base::ISOdatetime()`).
`ymd(x),ydm(x),mdy(x), myd(x),dmy(x),dym(x)`	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: `myd("June,2020,16")`
`XXX_hms(x),XXX_hm(x), XXX_h(x)`	Any of the above 6 parsing functions combined with `_hms,_hm, _h` parse date-times with year, month, and day, hour (**h**), minute (**m**), and second (**s**) components. Example: `dmy_hm("31/05/2020 1:02")`

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:

`x <- as_datetime("2020-05-28 14:00:00 UTC")`	
`wday(x, label=TRUE)`	Extracts the day of the week (here by name).
`tz(x) <- "America/New_York"`	This replaces the UTC time zone.

Time operations

`interval(start, end)` or `start %--% end`	Create an *Interval* object with the specified start and end dates. Example: `dmy("10/05/2020") %--% dmy("17/05/2020")`
`int_length(int)`	Return the length of the interval in seconds.
`duration(num, units)`	Create a *duration* object (=exact time measurements); default is in seconds.
`as.duration(x,..)`	Coerce a time interval into a *duration* object. Example: `as.duration(day_int)` Similar to `int_length(int)`
`make_difftime(num, units)`	Create a *difftime* object with the specified number of units.

Strings

Tidyverse approach to string manipulation

`load(stringr)` or `load(tidyverse)`

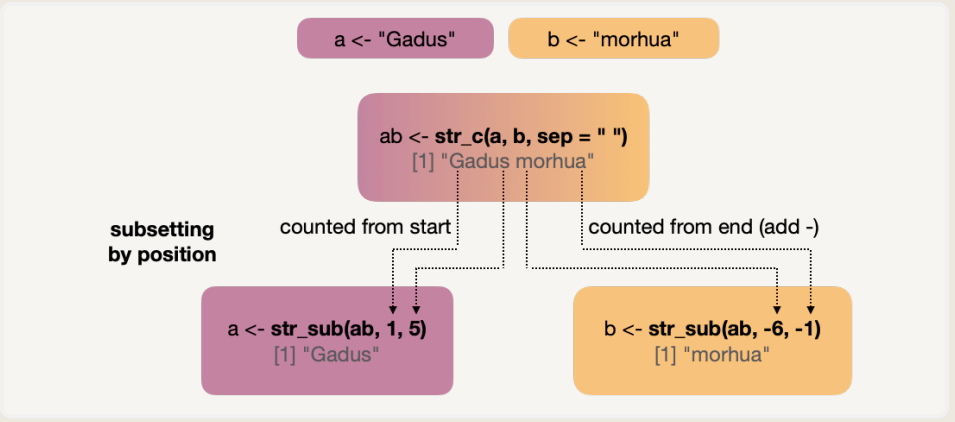
Some stringr functions have an equivalent base function:

Check & modify strings

stringr functions	Description	Base functions
`str_length(string)`	Number of characters in string.	`nchar(x)`
`str_to_upper(string)`	Convert to upper case.	`toupper(x)`
`str_to_lower(string)`	Convert to lower case.	`tolower(x)`
`str_to_title(string)`	Convert to title case.	
`str_trim(string, side)`	Trim whitespace from the start and/or end of a string.	
`str_pad(string, width)`	Pad strings to constant width.	
`str_trunc(string, width)`	Truncate the width of strings.	

Combine

stringr functions	Description	Base functions
`str_c(x,y, sep=' ')`	Join multiple vectors together.	`paste(x,y, sep=' ')`
`str_c(x, collapse=' ')`	Join vector elements together.	`paste(x, collapse=' ')`



Order strings

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

Detect & locate patterns

stringr functions	Description	Base functions
`str_detect(string, pattern)`	Detect patterns in string, returns logical vector.	`grepl(pattern, x)`
`str_which(string,pattern)`	Return position of strings that contain a pattern match (wrapper for *which(str_detect(string, pattern)*.	`grep(pattern, x)`
`str_count(string,pattern)`	Count the number of matches in a string.	
`str_locate(string, pattern)`	Find starting and end position of first match in each string, output is a matrix.	
`str_locate_all(string, pattern)`	Find starting and end position of all matches within each string, output is a list.	

Extract & subset patterns

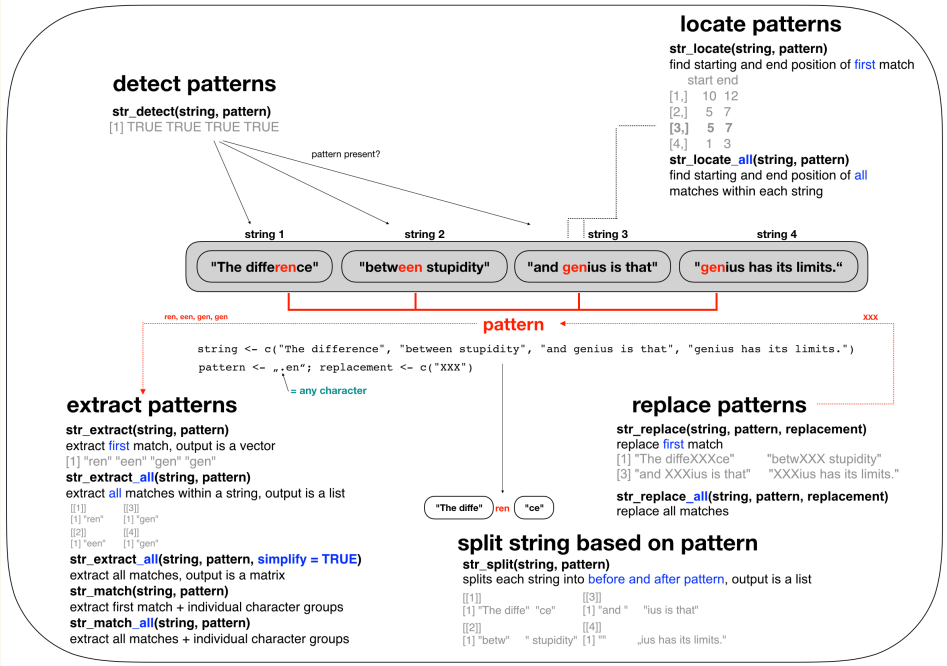
stringr functions	Description	Base functions
`str_sub(string, start, end)`	Extract and replace substrings at a specific position from a character vector.	`substr(x,start,stop)`
`str_extract(string, pattern)`	Return the first pattern match found in each string, as a vector.	
`str_extract_all(string, pattern,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

Replace patterns

stringr functions	Description	Base functions
`str_replace(pattern, replacement, x)`	Perform replacement of first match.	`sub(pattern, replacement, x)`
`str_replace_all(pattern, replacement, x)`	Perform replacement of all matches.	`gsub(pattern, replacement, x)`

Split based on patterns

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



Regular expressions

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

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

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

Special Metacharacters	
\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\f	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.