essencemediacom

Wprowadzenie do Tidyverse

02
03
23

Część II

Co dzisiaj

Operacje na zmiennych

Operacje na wierszach

Pivoty danych

Praca z tekstem

Operacje na zmiennych

Biblioteki w ramach Tidyverse'a

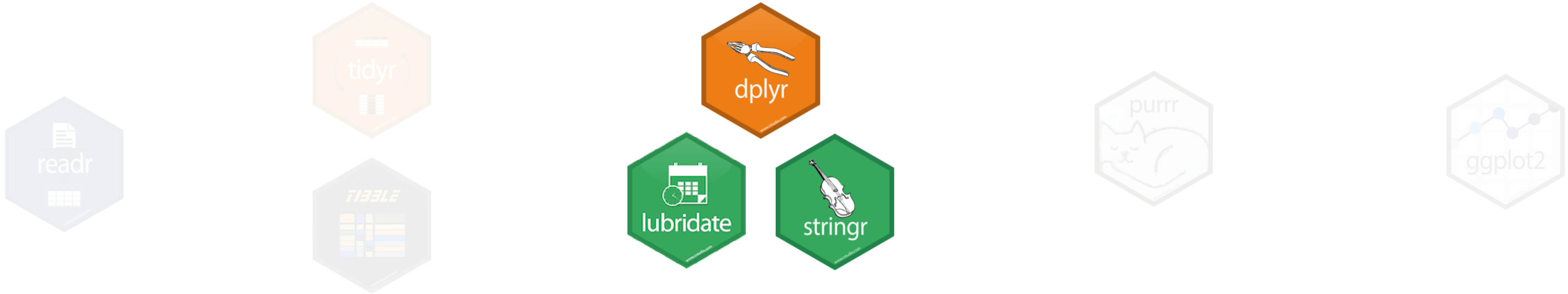
Import danych

Uporządkowanie danych

Przetwarzanie danych

Programowanie

Wizualizacja



Przetwarzanie danych z pomocą biblioteki dplyr

Uporządkowany zbiór czasowników

Czasownik	Opis
group_by()	grupuje na podstawie zadanych zmiennych
summarise()	na podstawie group_by tworzy agregaty zmiennych
mutate()	dodaje nową zmienną przy pomocy dodatkowych funkcji
select()	wybiera zmienną na podstawie jej nazwy
rename()	zmienia nazwy kolumn
filter()	filtruje zmienne na podstawie ich wartości
arrange()	zmienia kolejność wierszy

Operacje na wierszach

Biblioteki w ramach Tidyverse'a

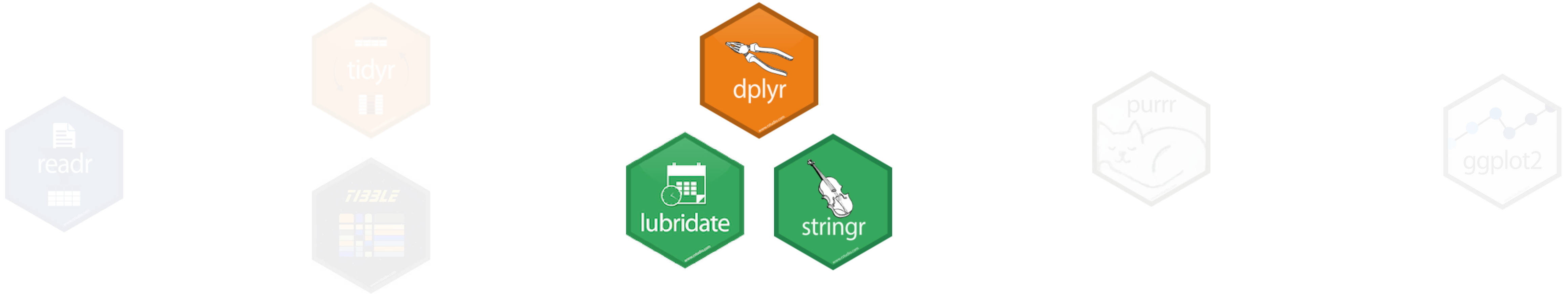
Import danych

Uporządkowanie danych

Przetwarzanie danych

Programowanie

Wizualizacja



Przetwarzanie danych z pomocą biblioteki dplyr

Uporządkowany zbiór czasowników

Czasownik	Opis
group_by()	grupuje na podstawie zadanych zmiennych
summarise()	na podstawie group_by tworzy agregaty zmiennych
mutate()	dodaje nową zmienną przy pomocy dodatkowych funkcji
select()	wybiera zmienną na podstawie jej nazwy
rename()	zmienia nazwy kolumn
filter()	filtruje zmienne na podstawie ich wartości
arrange()	zmienia kolejność wierszy

Pivotowanie danych

Biblioteki w ramach Tidyverse'a

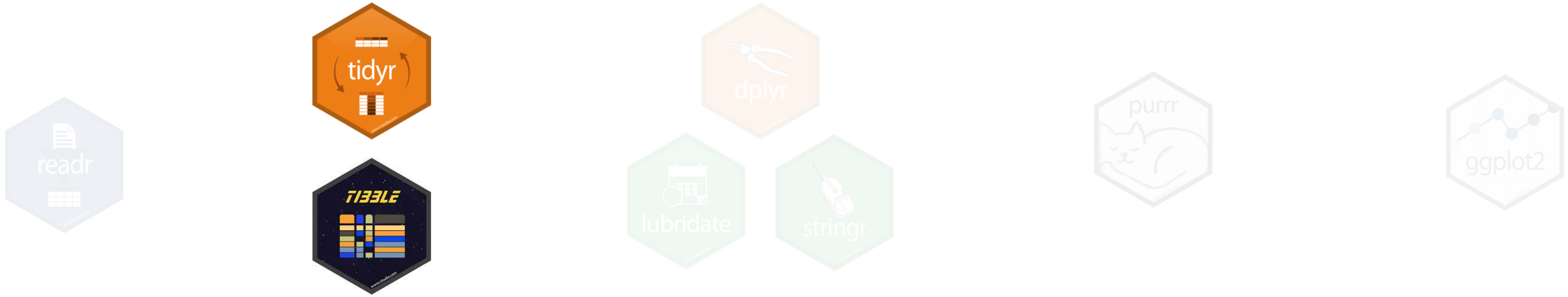
Import danych

Uporządkowanie danych

Przetwarzanie danych

Programowanie

Wizualizacja

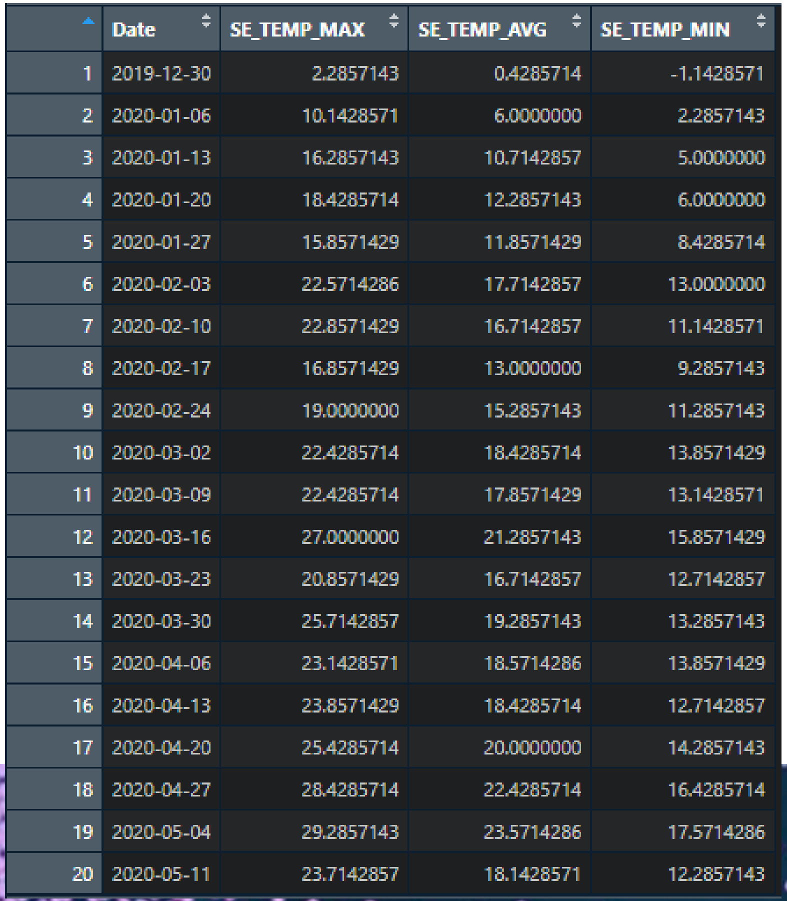


Tworzenie uporządkowanego zbioru danych

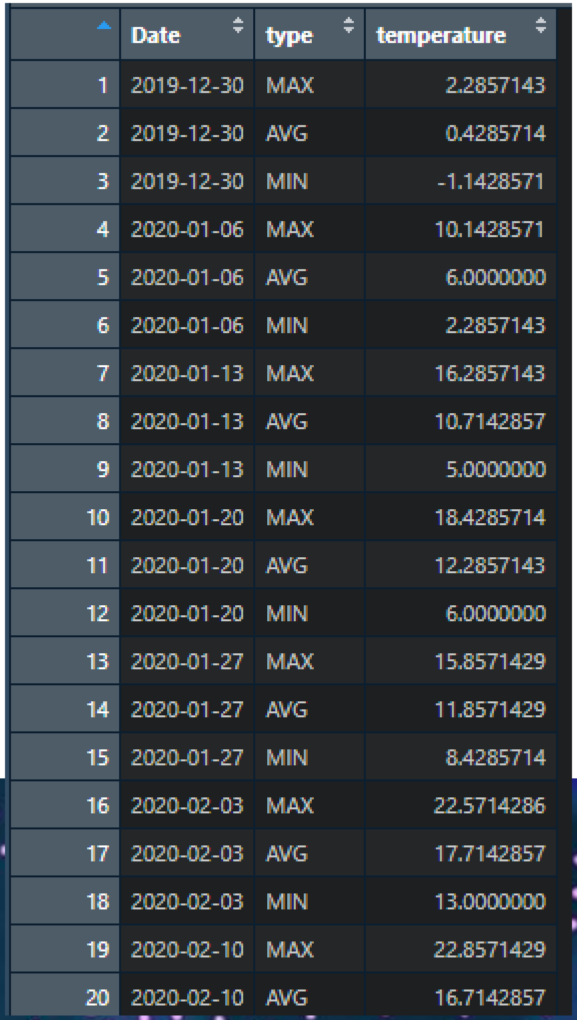
Funkcje w ramach biblioteki tidyr

Funkcja	Opis
separate()	Oddziela / dzieli pojedynczą kolumnę na wiele kolumn
unite()	W przeciwieństwie do separate() - łączy dwie lub więcej kolumn w jedną
pivot_wider()	Funkcja „rozszerza" wiele kolumn ze zbioru danych i konwertuje je na pary klucz-wartość — POZIOMO
pivot_longer()	Funkcja zajmuje dwie kolumny i „wydłuża" je w kilka kolumn - PIONOWO

Dane długie (long) a dane szerokie (wide)



	Date	SE_TEMP_MAX	SE_TEMP_AVG	SE_TEMP_MIN
1	2019-12-30	2.2857143	0.4285714	-1.1428571
2	2020-01-06	10.1428571	6.0000000	2.2857143
3	2020-01-13	16.2857143	10.7142857	5.0000000
4	2020-01-20	18.4285714	12.2857143	6.0000000
5	2020-01-27	15.8571429	11.8571429	8.4285714
6	2020-02-03	22.5714286	17.7142857	13.0000000
7	2020-02-10	22.8571429	16.7142857	11.1428571
8	2020-02-17	16.8571429	13.0000000	9.2857143
9	2020-02-24	19.0000000	15.2857143	11.2857143
10	2020-03-02	22.4285714	18.4285714	13.8571429
11	2020-03-09	22.4285714	17.8571429	13.1428571
12	2020-03-16	27.0000000	21.2857143	15.8571429
13	2020-03-23	20.8571429	16.7142857	12.7142857
14	2020-03-30	25.7142857	19.2857143	13.2857143
15	2020-04-06	23.1428571	18.5714286	13.8571429
16	2020-04-13	23.8571429	18.4285714	12.7142857
17	2020-04-20	25.4285714	20.0000000	14.2857143
18	2020-04-27	28.4285714	22.4285714	16.4285714
19	2020-05-04	29.2857143	23.5714286	17.5714286
20	2020-05-11	23.7142857	18.1428571	12.2857143



	Date	type	temperature
1	2019-12-30	MAX	2.2857143
2	2019-12-30	AVG	0.4285714
3	2019-12-30	MIN	-1.1428571
4	2020-01-06	MAX	10.1428571
5	2020-01-06	AVG	6.0000000
6	2020-01-06	MIN	2.2857143
7	2020-01-13	MAX	16.2857143
8	2020-01-13	AVG	10.7142857
9	2020-01-13	MIN	5.0000000
10	2020-01-20	MAX	18.4285714
11	2020-01-20	AVG	12.2857143
12	2020-01-20	MIN	6.0000000
13	2020-01-27	MAX	15.8571429
14	2020-01-27	AVG	11.8571429
15	2020-01-27	MIN	8.4285714
16	2020-02-03	MAX	22.5714286
17	2020-02-03	AVG	17.7142857
18	2020-02-03	MIN	13.0000000
19	2020-02-10	MAX	22.8571429
20	2020-02-10	AVG	16.7142857

Do modelowania potrzeba danych **szerokich**

Ale dane **długie** są łatwiejsze do obróbki, wygodniejsze do trzymania w bazach danych oraz wymagane przez niektóre programy.

Praca z tekstem

Biblioteki w ramach Tidyverse'a

Import danych

readr

Uporządkowanie danych

tidyr

TIBBLE

Przetwarzanie danych

dplyr

lubridate

stringr

Programowanie

purrr

Wizualizacja

ggplot2

Praca z tekstem

Funkcje w ramach pakietu stringr

Funkcja	Opis
str_length()	Długość wyrażenia tekstowego
str_c()	Sklejenie kilku wyrażeń tekstowych

str_replace()
str_replace_all()
	Podmiana fragmentu tekstu

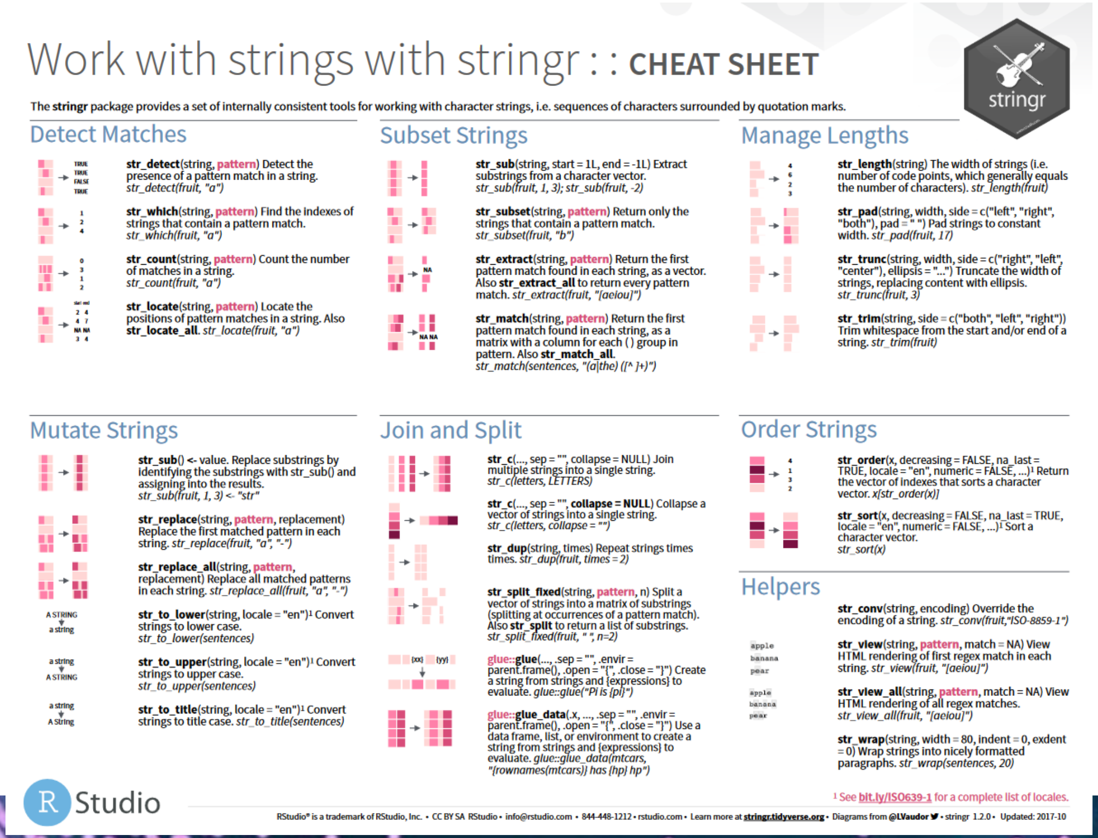
str_detect()	Sprawdzenie występowania w jednym tekście innego tekstu

Wyrażenia regularne – wstęp do wstępu

Używanie wyrażeń regularnych

Wyrażenie	Co to jest
a	Litera ☺
\	\
\”	”
[:upper:]	Wielkie litery
[:punct:]	Znaki przestankowe
… i inne	

StringR cheat sheet


TRUE
TRUE
FALSE
TRUE
1
2
4
0
3
1
2
start end
2 4
4 7
NA NA
3 4
NA
NA NA
4
6
2
3
A STRING
a string
a string
A STRING
a string
A String
{xx} {yy}
4
1
3
2
apple
banana
pear
apple
banana
pear
stringr

Work with strings with stringr : : CHEAT SHEET

The **stringr** package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks.

Detect Matches

str_detect(string, pattern) Detect the presence of a pattern match in a string. *str_detect(fruit, "a")*

str_which(string, pattern) Find the indexes of strings that contain a pattern match. *str_which(fruit, "a")*

str_count(string, pattern) Count the number of matches in a string. *str_count(fruit, "a")*

str_locate(string, pattern) Locate the positions of pattern matches in a string. Also **str_locate_all**. *str_locate(fruit, "a")*

Subset Strings

str_sub(string, start = 1L, end = -1L) Extract substrings from a character vector. *str_sub(fruit, 1, 3); str_sub(fruit, -2)*

str_subset(string, pattern) Return only the strings that contain a pattern match. *str_subset(fruit, "b")*

str_extract(string, pattern) Return the first pattern match found in each string, as a vector. Also **str_extract_all** to return every pattern match. *str_extract(fruit, "[aeiou]")*

str_match(string, pattern) Return the first pattern match found in each string, as a matrix with a column for each ( ) group in pattern. Also **str_match_all**. *str_match(sentences, "(a|the) ([^]+)")*

Manage Lengths

str_length(string) The width of strings (i.e. number of code points, which generally equals the number of characters). *str_length(fruit)*

str_pad(string, width, side = c("left", "right", "both"), pad = " ") Pad strings to constant width. *str_pad(fruit, 17)*

str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...") Truncate the width of strings, replacing content with ellipsis. *str_trunc(fruit, 3)*

str_trim(string, side = c("both", "left", "right")) Trim whitespace from the start and/or end of a string. *str_trim(fruit)*

Mutate Strings

str_sub() <- value. Replace substrings by identifying the substrings with str_sub() and assigning into the results. *str_sub(fruit, 1, 3) <- "str"*

str_replace(string, pattern, replacement) Replace the first matched pattern in each string. *str_replace(fruit, "a", "-")*

str_replace_all(string, pattern, replacement) Replace all matched patterns in each string. *str_replace_all(fruit, "a", "-")*

str_to_lower(string, locale = "en")[1] Convert strings to lower case. *str_to_lower(sentences)*

str_to_upper(string, locale = "en")[1] Convert strings to upper case. *str_to_upper(sentences)*

str_to_title(string, locale = "en")[1] Convert strings to title case. *str_to_title(sentences)*

Join and Split

str_c(..., sep = "", collapse = NULL) Join multiple strings into a single string. *str_c(letters, LETTERS)*

str_c(..., sep = "", **collapse = NULL**) Collapse a vector of strings into a single string. *str_c(letters, collapse = "")*

str_dup(string, times) Repeat strings times times. *str_dup(fruit, times = 2)*

str_split_fixed(string, pattern, n) Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also **str_split** to return a list of substrings. *str_split_fixed(fruit, " ", n=2)*

glue::**glue**(..., .sep = "", .envir = parent.frame(), .open = "{", .close = "}") Create a string from strings and {expressions} to evaluate. *glue::glue("Pi is {pi}")*

glue::**glue_data**(.x, ..., .sep = "", .envir = parent.frame(), .open = "{", .close = "}") Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate. *glue::glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")*

Order Strings

str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Return the vector of indexes that sorts a character vector. *x[str_order(x)]*

str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Sort a character vector. *str_sort(x)*

Helpers

str_conv(string, encoding) Override the encoding of a string. *str_conv(fruit,"ISO-8859-1")*

str_view(string, pattern, match = NA) View HTML rendering of first regex match in each string. *str_view(fruit, "[aeiou]")*

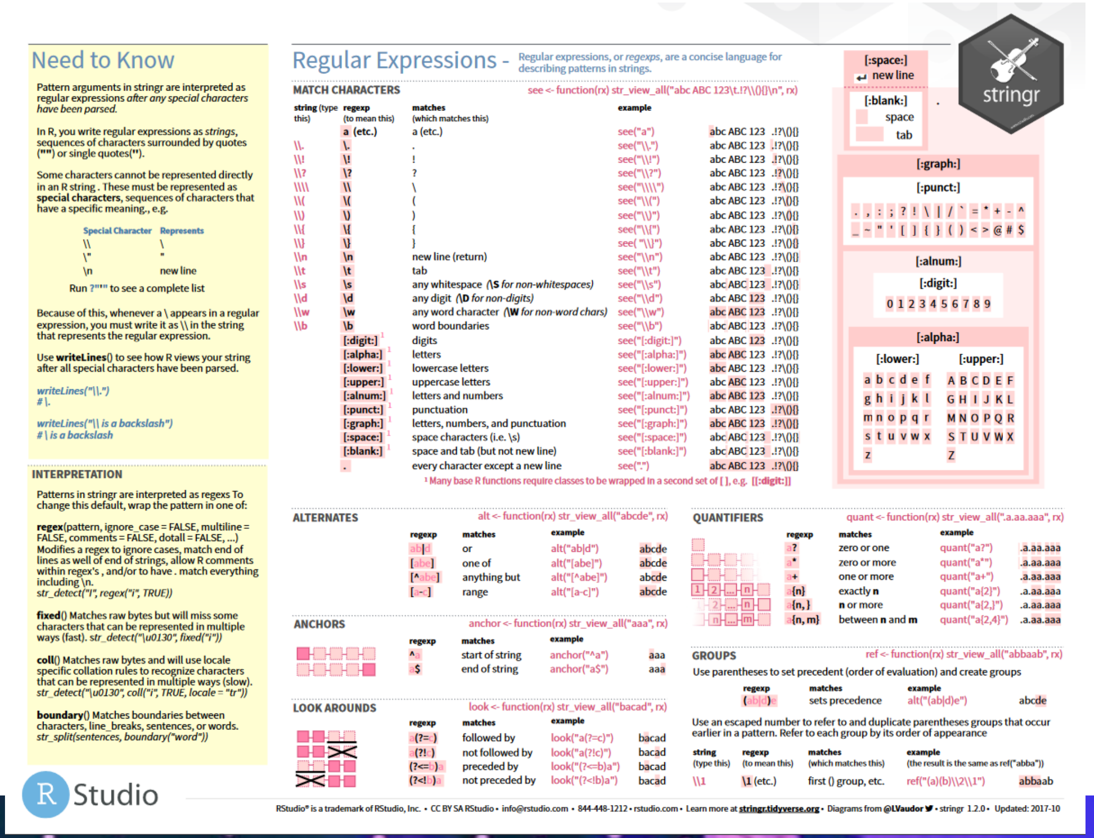
str_view_all(string, pattern, match = NA) View HTML rendering of all regex matches. *str_view_all(fruit, "[aeiou]")*

str_wrap(string, width = 80, indent = 0, exdent = 0) Wrap strings into nicely formatted paragraphs. *str_wrap(sentences, 20)*

[1] See bit.ly/ISO639-1 for a complete list of locales.

R Studio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **stringr.tidyverse.org** • Diagrams from **@LVaudor** • stringr 1.2.0 • Updated: 2017-10



Need to Know

Pattern arguments in stringr are interpreted as regular expressions *after any special characters have been parsed.*

In R, you write regular expressions as *strings*, sequences of characters surrounded by quotes ("") or single quotes('').

Some characters cannot be represented directly in an R string . These must be represented as **special characters**, sequences of characters that have a specific meaning., e.g.

Special Character	Represents
\\	\
\"	"
\n	new line

Run ?"'" to see a complete list

Because of this, whenever a \ appears in a regular expression, you must write it as \\ in the string that represents the regular expression.

Use **writeLines**() to see how R views your string after all special characters have been parsed.

writeLines("\\.")
# \.

writeLines("\\ is a backslash")
# \ is a backslash

INTERPRETATION

Patterns in stringr are interpreted as regexs To change this default, wrap the pattern in one of:

regex(pattern, ignore_case = FALSE, multiline = FALSE, comments = FALSE, dotall = FALSE, ...) Modifies a regex to ignore cases, match end of lines as well of end of strings, allow R comments within regex's , and/or to have . match everything including \n. *str_detect("I", regex("i", TRUE))*

fixed() Matches raw bytes but will miss some characters that can be represented in multiple ways (fast). *str_detect("\u0130", fixed("i"))*

coll() Matches raw bytes and will use locale specific collation rules to recognize characters that can be represented in multiple ways (slow). *str_detect("\u0130", coll("i", TRUE, locale = "tr"))*

boundary() Matches boundaries between characters, line_breaks, sentences, or words. *str_split(sentences, boundary("word"))*

Regular Expressions - Regular expressions, or *regexps*, are a concise language for describing patterns in strings.

MATCH CHARACTERS

see <- function(rx) str_view_all("abc ABC 123\t.!?\\(){}\n", rx)

string (type this)	regexp (to mean this)	matches (which matches this)	example	
	a (etc.)	a (etc.)	see("a")	abc ABC 123 .!?\(){}
\\.	\.	.	see("\\.")	abc ABC 123 .!?\(){}
\\!	\!	!	see("\\!")	abc ABC 123 .!?\(){}
\\?	\?	?	see("\\?")	abc ABC 123 .!?\(){}
\\\\	\\	\	see("\\\\")	abc ABC 123 .!?\(){}
\\(	\(	(	see("\\(")	abc ABC 123 .!?\(){}
\\)	\)	)	see("\\)")	abc ABC 123 .!?\(){}
\\{	\{	{	see("\\{")	abc ABC 123 .!?\(){}
\\}	\}	}	see("\\}")	abc ABC 123 .!?\(){}
\\n	\n	new line (return)	see("\\n")	abc ABC 123 .!?\(){}
\\t	\t	tab	see("\\t")	abc ABC 123 .!?\(){}
\\s	\s	any whitespace (\S *for non-whitespaces)*	see("\\s")	abc ABC 123 .!?\(){}
\\d	\d	any digit (\D *for non-digits)*	see("\\d")	abc ABC 123 .!?\(){}
\\w	\w	any word character (\W *for non-word chars)*	see("\\w")	abc ABC 123 .!?\(){}
\\b	\b	word boundaries	see("\\b")	abc ABC 123 .!?\(){}
	[:digit:][1]	digits	see("[:digit:]")	abc ABC 123 .!?\(){}
	[:alpha:][1]	letters	see("[:alpha:]")	abc ABC 123 .!?\(){}
	[:lower:][1]	lowercase letters	see("[:lower:]")	abc ABC 123 .!?\(){}
	[:upper:][1]	uppercase letters	see("[:upper:]")	abc ABC 123 .!?\(){}
	[:alnum:][1]	letters and numbers	see("[:alnum:]")	abc ABC 123 .!?\(){}
	[:punct:][1]	punctuation	see("[:punct:]")	abc ABC 123 .!?\(){}
	[:graph:][1]	letters, numbers, and punctuation	see("[:graph:]")	abc ABC 123 .!?\(){}
	[:space:][1]	space characters (i.e. \s)	see("[:space:]")	abc ABC 123 .!?\(){}
	[:blank:][1]	space and tab (but not new line)	see("[:blank:]")	abc ABC 123 .!?\(){}
	.	every character except a new line	see(".")	abc ABC 123 .!?\(){}

[1] Many base R functions require classes to be wrapped in a second set of [ ], e.g. **[[:digit:]]**

[:space:]
↵ new line
[:blank:]
space
tab
.

[:graph:]
[:punct:]
. , : ; ? ! \ | / ` = * + - ^
_ ~ " ' [ ] { } ( ) < > @ # $

[:alnum:]
[:digit:]
0 1 2 3 4 5 6 7 8 9

[:alpha:]
[:lower:] [:upper:]
a b c d e f A B C D E F
g h i j k l G H I J K L
m n o p q r M N O P Q R
s t u v w x S T U V W X
z Z

ALTERNATES

alt <- function(rx) str_view_all("abcde", rx)

regexp	matches	example	
ab|d	or	alt("ab|d")	abcde
[abe]	one of	alt("[abe]")	abcde
[^abe]	anything but	alt("[^abe]")	abcde
[a-c]	range	alt("[a-c]")	abcde

ANCHORS

anchor <- function(rx) str_view_all("aaa", rx)

regexp	matches	example	
^a	start of string	anchor("^a")	aaa
a$	end of string	anchor("a$")	aaa

LOOK AROUNDS

look <- function(rx) str_view_all("bacad", rx)

regexp	matches	example	
a(?=c)	followed by	look("a(?=c)")	bacad
a(?!c)	not followed by	look("a(?!c)")	bacad
(?<=b)a	preceded by	look("(?<=b)a")	bacad
(?<!b)a	not preceded by	look("(?<!b)a")	bacad

QUANTIFIERS

quant <- function(rx) str_view_all(".a.aa.aaa", rx)

regexp	matches	example	
a?	zero or one	quant("a?")	.a.aa.aaa
a*	zero or more	quant("a*")	.a.aa.aaa
a+	one or more	quant("a+")	.a.aa.aaa
a{n}	exactly **n**	quant("a{2}")	.a.aa.aaa
a{n, }	**n** or more	quant("a{2,}")	.a.aa.aaa
a{n, m}	between **n** and **m**	quant("a{2,4}")	.a.aa.aaa

GROUPS

ref <- function(rx) str_view_all("abbaab", rx)

Use parentheses to set precedent (order of evaluation) and create groups

regexp	matches	example	
(ab|d)e	sets precedence	alt("(ab|d)e")	abcde

Use an escaped number to refer to and duplicate parentheses groups that occur earlier in a pattern. Refer to each group by its order of appearance

string (type this)	regexp (to mean this)	matches (which matches this)	example (the result is the same as ref("abba"))	
\\1	\1 (etc.)	first () group, etc.	ref("(a)(b)\\2\\1")	abbaab

R Studio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **stringr.tidyverse.org** • Diagrams from **@LVaudor** • stringr 1.2.0 • Updated: 2017-10

essence**mediacom**
business science

Jarek Dejneka, Managing Partner
jaroslaw.dejneka@essencemediacom.com

Bartek Kowalski, Business Science Director
bartosz.kowalski@essencemediacom.com

@ mbs@mediacom.com

@MBSWarsaw

@MBSWarsaw

business-science.pl

„An investment in knowledge always pays the best interest”

Benjamin Franklin