WHAT ARE REGULAR EXPRESSIONS?

- Regular expressions are powerful tools to work with data

- String sequence that is used as a pattern to search textual data
- Eg.: `"cent[er]{2}"`

- Packages you need:
  - `stringr` to work with strings and regular expressions
  - `lubridate` to work with dates

HOW DOES IT WORK?

- Regular expressions are written as any other strings "" or ‘’

- By default, the `stringr` package passes every string as regex
- In case it does not, wrap the string in `regex()`

STRING PATTERN REPLACEMENT

```
> str_replace("Unviersity", "unvi", "univ")
[1] "Unviersity"
> str_replace("Unviersity", regex("unvi", ignore_case = TRUE), "univ")
[1] "university"
```

ESCAPE SPECIAL CHARACTERS

- Special characters exist and need to be escaped by \

TYPE THIS	MEAN THIS	MATCH THIS
\\?	\?	?

```
> q <- "What can you do with a materials engineering degree?"
> str_extract(q, "degree?")
[1] "degree"
> str_extract(q, "degree\\?")
[1] "degree?"
```

ESCAPE

MATCH PATTERN IN A STRING

- `str_detect(STRING, PATTERN)` detect the presence of a pattern
- `str_extract(...)` return the first pattern match in a string
- `str_extract_all(...)` return every pattern match in a string

- `str_count(...)` count the number of matches
- `str_subset(...)` return only the strings that contain a pattern

MUTATE A STRING

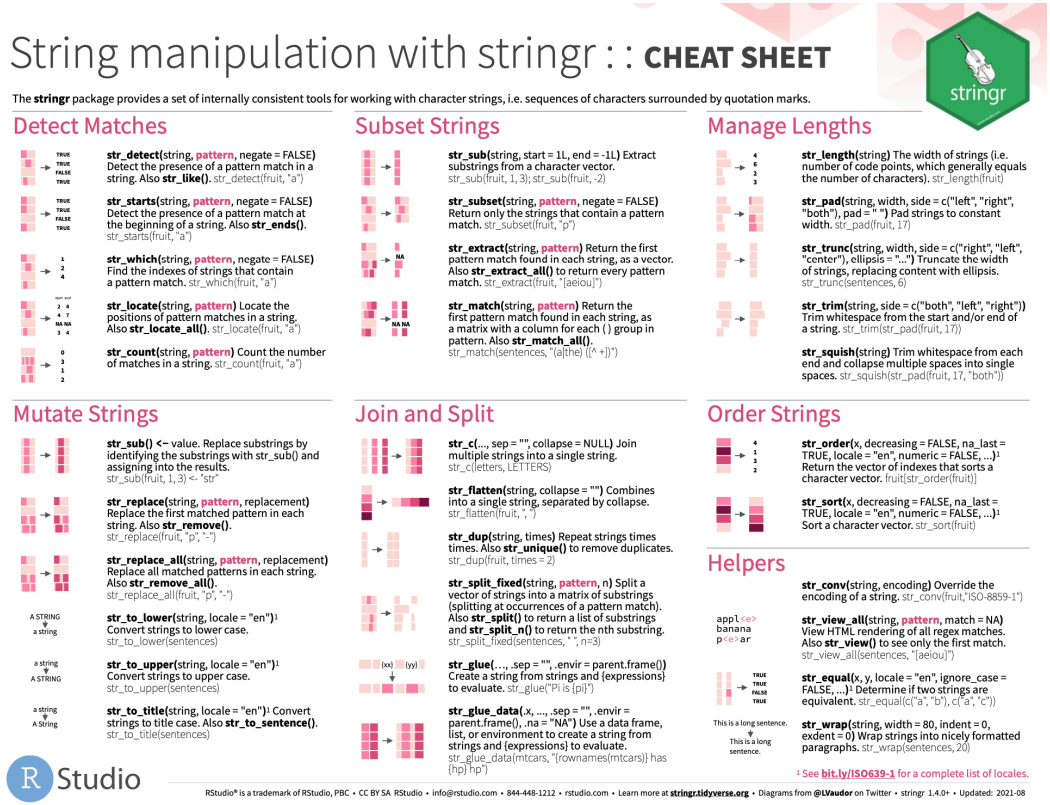
- str_replace(STRING, PATTERN, REPLACEMENT)
- str_replace_all(...)

- str_remove(STRING, PATTERN)
- str_remove_all(...)

- str_to_lower(STRING)
- str_to_upper(...)
- str_conv(STRING, ENCODING)

JOIN OR SPLIT A STRING

- `str_c(STRING1, STRING2, SEP = "")` join multiple strings in a single string
- `str_flatten(STRING, COLLAPSE = "")` combines all strings into one
- `str_dup(STRING, TIMES)` duplicates strings



String manipulation 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.

stringr

Detect Matches

str_detect(string, **pattern**, negate = FALSE**)** Detect the presence of a pattern match in a string. Also **str_like()**. str_detect(fruit, "a")

str_starts(string, **pattern**, negate = FALSE**)** Detect the presence of a pattern match at the beginning of a string. Also **str_ends()**. str_starts(fruit, "a")

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

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

str_count(string, **pattern**) Count the number of matches in a string. str_count(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**, negate = FALSE**)** Return only the strings that contain a pattern match. str_subset(fruit, "p")

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(sentences, 6)

str_trim(string, side = c("both", "left", "right"**)** Trim whitespace from the start and/or end of a string. str_trim(str_pad(fruit, 17))

str_squish(string**)** Trim whitespace from each end and collapse multiple spaces into single spaces. str_squish(str_pad(fruit, 17, "both"))

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. Also **str_remove()**. str_replace(fruit, "p", "-")

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

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. Also **str_to_sentence()**. str_to_title(sentences)

Join and Split

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

str_flatten(string, collapse = ""**)** Combines into a single string, separated by collapse. str_flatten(fruit, ", ")

str_dup(string, times**)** Repeat strings times times. Also **str_unique()** to remove duplicates. 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 and **str_split_n()** to return the nth substring. str_split_fixed(sentences, " ", n=3)

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

str_glue_data(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA"**)** Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate. str_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. fruit[str_order(fruit)]

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

Helpers

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

str_view_all(string, **pattern**, match = NA**)** View HTML rendering of all regex matches. Also **str_view()** to see only the first match. str_view_all(sentences, "[aeiou]")

str_equal(x, y, locale = "en", ignore_case = FALSE, ...**)**[1] Determine if two strings are equivalent. str_equal(c("a", "b"), c("a", "c"))

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.

RStudio

CREATE A NEW VARIABLE

- Create a new variable based on the content of a character variable
- use piping in `tidyverse` to iterate over each observation

  %>%

- E.g. Twitter data
- create a new variable about the user: *institution* or *community*
- based on the information in
  - username
  - user description
  - hyperlink

NEW SOCIAL VARIABLE: INSTITUTION

```
institutions <- regex("university|college|academy|school|center|centre|departme

df2 <- df %>%
  unite("united_names", name, screen_name, description, profile_expanded_url,
        sep = " ", remove = FALSE) %>%
  mutate(institution = str_detect(united_names, institutions))
```

NEW VARIABLE

STRING

PATTERN

NEW VARIABLE

united_names	institution
Sharon Schladow collegepathSS Consultant since 2001 visiting 35~50 campuses/yr focus on…	TRUE
MontclareLabs MontclareLabs The lab focuses on protein engineering & molecular design. L…	TRUE
JayLite9 JayLite9 I invest in real estate, speak fluent sarcasm, question as much as possible, …	FALSE

NEW LEXICAL VARIABLE: SPELLING

NEW VARIABLE

STRING

PATTERN

```
df4 <- df3 %>%
  mutate(center = str_extract(text, "center|centre"),
    center = as.factor(center))
```

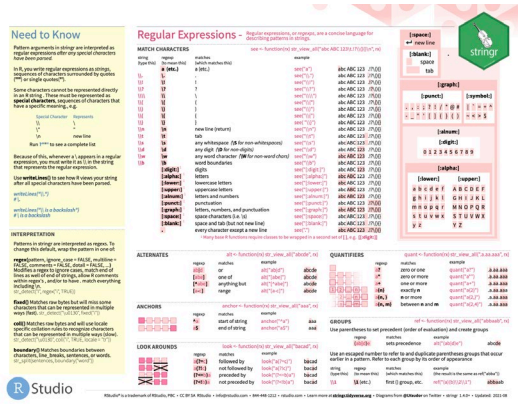
STRING

NEW VARIABLE

text	center
I'm a well-known academ c center of scholarly excellence Place your Oder today today #Math #accountin…	center
Over 26,000 students to write Telangana Engineering CET tomorrow at 96 centres - The New Indian Expre…	centre
Software Engineer in Wageningen https://t.co/Qt5Ygxvsk6 #expat #jobs by Wageningen UR (University &a…	centre
Sports legends help the #CityofCarson get an upgrade! The Carol Kimmelman Athletic and Academic Cent…	center

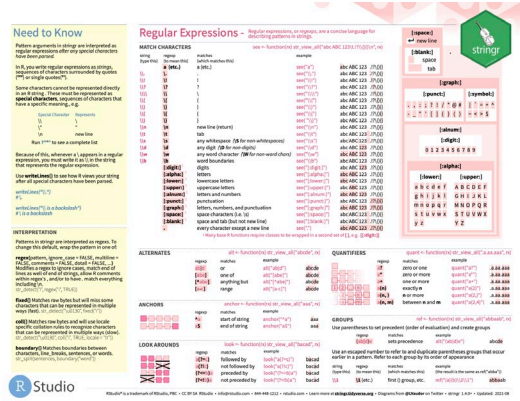
BUILDING A REGULAR EXPRESSION

TYPE	MEAN	REGEXP	MATCHES
|	OR	"center|centre"	"center" "centre"
[]	ANY OF	"cent[er]"	"cente" "centr"
()	GROUP	"cent(er|re)"	"center" "centre"
{min,max}	# OF TIMES	"cent[er]{2}"	"center" "centre"
.	ANY CHAR.	"cent.."	"center" "centre" "centra"
?	ZERO or ONE	"center.?"	"center" "centers"
+	ONE or MORE	"cent.+"	"center" "centre" "centers"
*	ZERO or MORE	"cent.*"	"cent" "cents" "centre" "centers" "central"



USEFUL REGULAR EXPRESSIONS

REGEXP	MATCHES
`.`	any character
`[:space:]`	space
`[:punct:]`	punctuation marks
`[:digit:]`	numbers
`[:alpha:]`	letters
`[:alnum:]`	numbers and letters



NEW LINGUISTIC VARIABLE: CONTRACTIONS

Word	Contraction
are not	aren't
cannot	can't
could not	couldn't
did not	didn't
do not	don't
does not	doesn't
had not	hadn't
have not	haven't
he is	he's
he has	he's
he will	he'll

PREPARING THE REGULAR EXPRESSION

```
contractions <- read_excel("~/Dropbox/Work/ASEE 2020 Summer/ASEE-Cov
  mutate(Word = str_to_lower(Word),
         Contraction = str_to_lower(Contraction),
         cont = str_remove_all(Contraction, "[:space:]"),
         cont = str_replace_all(cont, "'|’|’", "['’’]"),
         notcont = str_replace_all(Word, "[:space:]", "[:space:]"))
```

Word	Contraction	cont	notcont
are not	aren’t	aren["’]t	are[:space:]not
cannot	can’t	can["’]t	cannot
could not	couldn’t	couldn["’]t	could[:space:]not
did not	didn’t	didn["’]t	did[:space:]not
do not	don’t	don["’]t	do[:space:]not
does not	doesn’t	doesn["’]t	does[:space:]not
had not	hadn’t	hadn["’]t	had[:space:]not
have not	haven’t	haven["’]t	have[:space:]not

PREPARING THE REGULAR EXPRESSION

SELECT COLUMN TO FLATTEN

COLLAPSE WITH | IN BETWEEN

```
cont_words <- str_flatten(contractions$cont, collapse = "|")
notcont_words <- str_flatten(contractions$notcont, collapse = "|")
cstr <- str_c(cont_words, notcont_words)
```

COMBINE THE TWO STRINGS

```
> cstr
[1] "aren['’]t|can['’]t|couldn['’]t|didn['’]t|don['’]t|doesn['’]t|hadn['’]
he['’]ll|he['’]d|here['’]s|i['’]m|i['’]ve|i['’]ll|i['’]d|i['’]d|isn['’]t
['’]ll|mustn['’]t|she['’]s|she['’]s|she['’]ll|she['’]d|she['’]d|shouldn['’
['’]re|they['’]ve|they['’]ll|they['’]d|they['’]d|wasn['’]t|we['’]re|we['’
weren['’]t|what['’]s|where['’]s|who['’]s|who['’]ll|won['’]t|wouldn['’]t|yo
ou['’]d|you['’]dare[:space:]not|cannot|could[:space:]not|did[:space:]not|do[:sp
pace:]not|have[:space:]not|he[:space:]is|he[:space:]has|he[:space:]will|he[:space
e:]am|i[:space:]have|i[:space:]will|i[:space:]would|i[:space:]had|is[:space:]not|
```

NEW LINGUISTIC VARIABLE: CONTRACTIONS

```
df3 <- df2 %>%
  select(name, description, institution, created_at, text) %>%
1: mutate(contraction = str_extract(text, regex(cstr, ignore_case = TRUE)),
          contraction = str_replace_all(contraction, "'|’|’", "'"),
          contraction = as.factor(str_to_lower(contraction)))

df4 <- df3 %>%
2: mutate(contr = case_when(
    str_detect(contraction, regex(cont_words)) ~ "contraction",
    str_detect(contraction, regex(notcont_words)) ~ "not contraction"),
    contr = as.factor(contr))
```

UNIFORM SPELLING

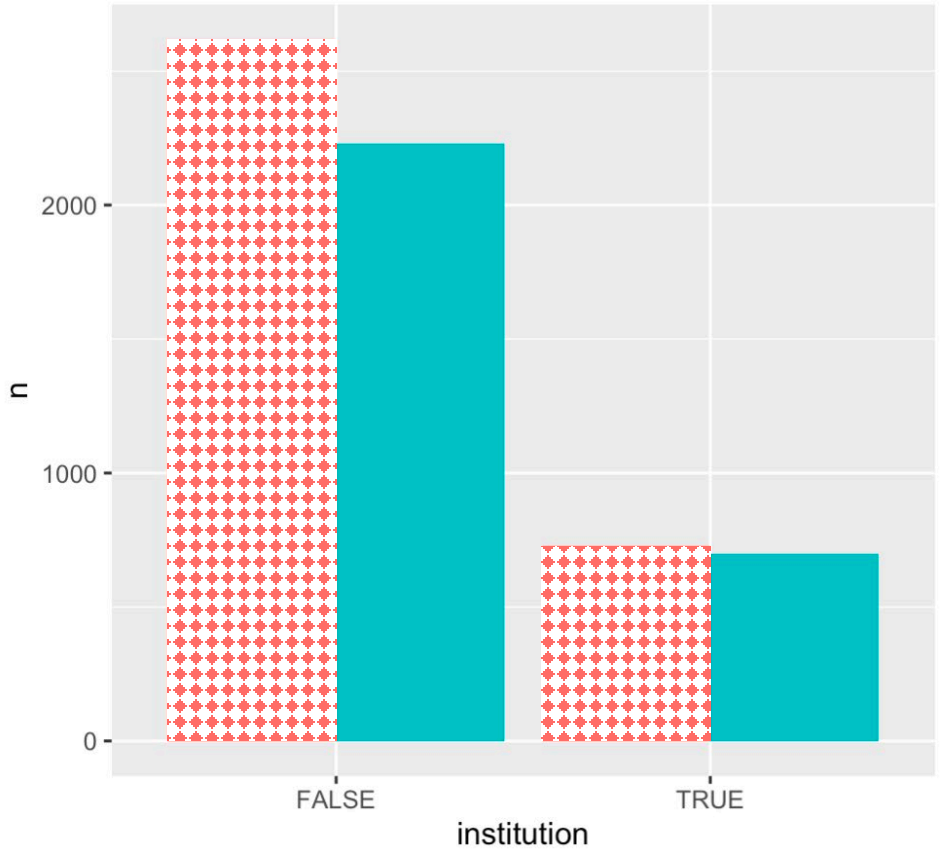
VECTORIZED *IF-ELSE* STATEMENT

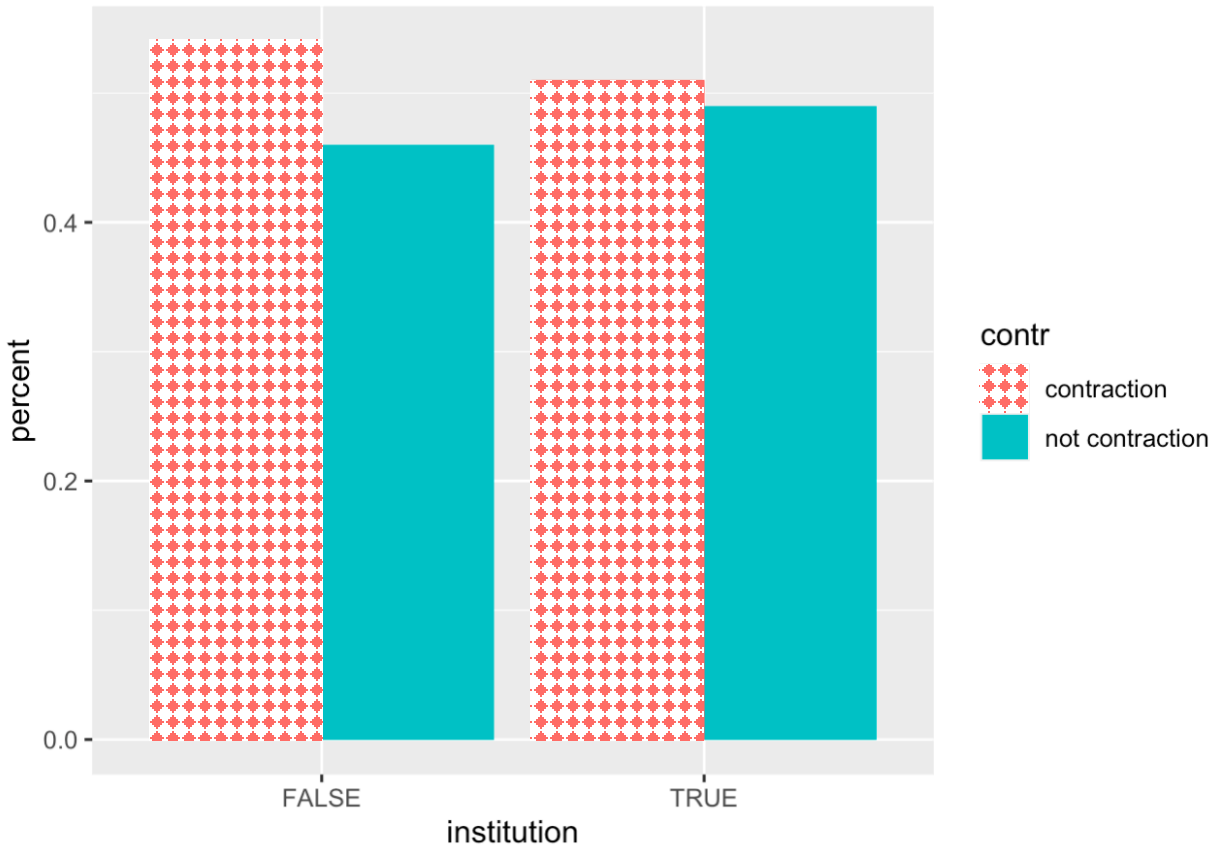
1: VARIABLE BASED ON EXTRACT CONTRACTION

2: VARIABLE BASED ON DETECT CONTRACTION TYPE

text	contraction	contr
Sound Transit is looking for a Program Manager–Risk …	it is	not contraction
@ScottAdamsSays There is a strict rule in test enginee…	there is	not contraction
@Slasher You need to read up on the reverse enginee…	don't	contraction
Simon Ang, a former University of Arkansas, Fayettevi…	*NA*	*NA*
JOB: Livonia MI USA – Laser Welding Process Engineer …	*NA*	*NA*

VISUALIZING CONTRACTIONS





FURTHER CONSIDERATIONS

- Compact regex is not always applicable

- E.g. *university, college, academy, research center*
  - very different forms require the OR statement: |
  - a list can be defined and used to match elements from it
  - `case_when()` function

- Special attention to the difference between `str_detect()` and `str_extract()`

- In an OR regex, longer argument will be evaluated first