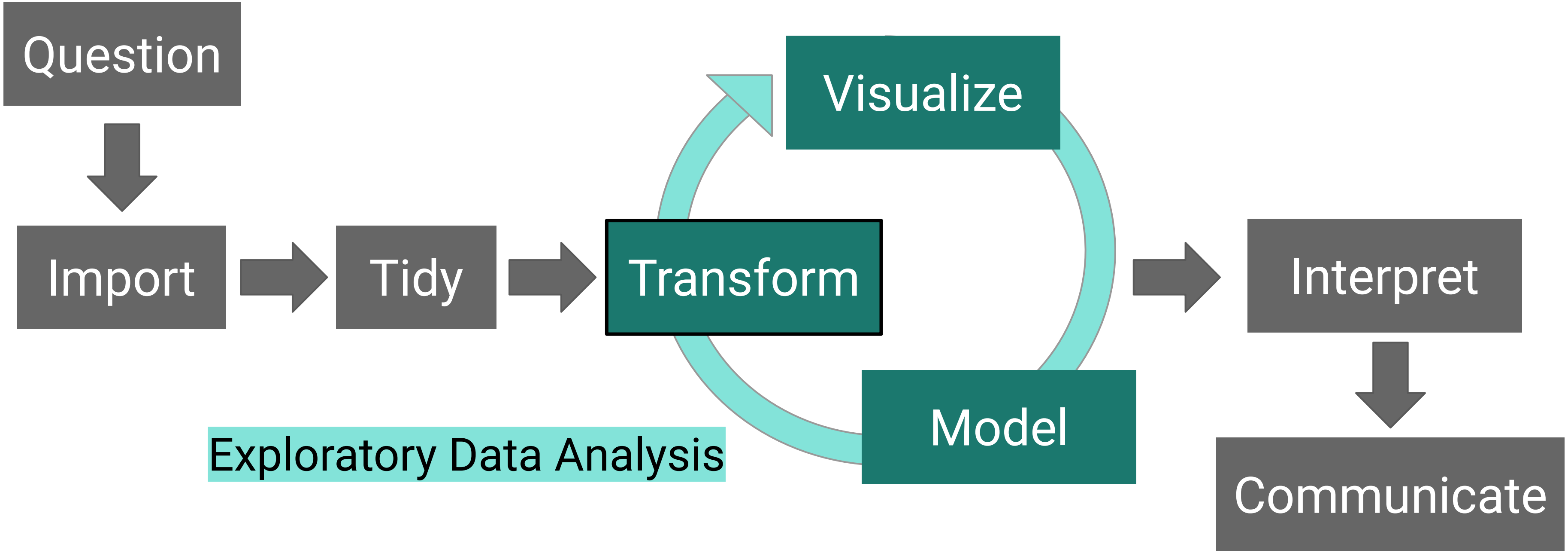
Strings

Lecture 8

Objective

- to manipulate strings using the **stringr** package

Motivation



Modified from https://r4ds.had.co.nz/explore-intro.html#explore-intro

Motivation

$154

3y2m

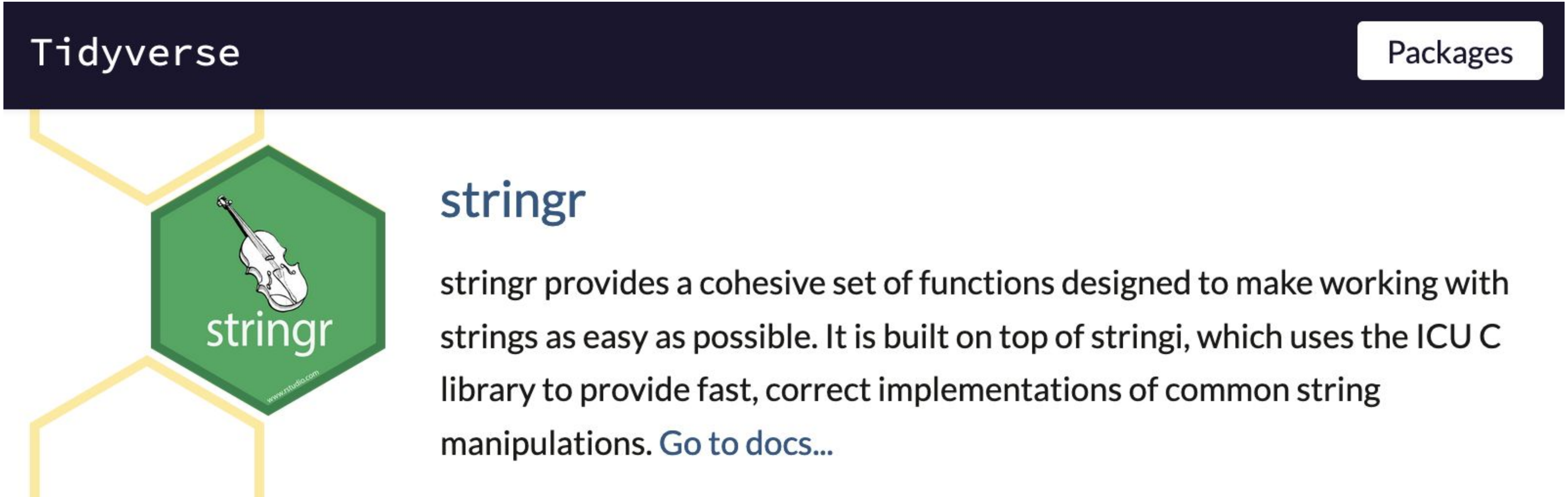
Country/Region

154

3 2

Country_Region

stringr package



stringr

stringr provides a cohesive set of functions designed to make working with strings as easy as possible. It is built on top of stringi, which uses the ICU C library to provide fast, correct implementations of common string manipulations. Go to docs...

https://www.tidyverse.org/packages/

String basics

- You can create strings with either single quotes or double quotes.

```
> string_1 <− "This sentence is a string."
> string_1
```

```
[1] "This sentence is a string."
```

- You can use special characters: "\n", newline, and "\t", tab.
- Use **writeLine( )** function to see desired output.

```
> string_2 <− "This is another sentence.\nAnd another one in a new line."
> string_2
```

```
> writeLine(string_2)
```

```
This is another sentence.
And another one in a new line.
```

String length

- Use **str_length( )** function to return the number of characters in a string.

```
> string_1 <− "This sentence is a string."
> str_lenght(string_1)
```

```
[1] 26
```

```
> str_length(c("one", "two and three", NA))
```

```
[1] 3 13 NA
```

Combining strings

- Use **str_c( )** function to combine two or more strings.
- The **sep** argument is used to control how the strings are separated.

```
> str_c("2021", "Feb", "15", sep = "-")
```

```
[1] "2021-Feb-15"
```

- Use the **collapse** argument to collapse a vector of strings into a single string

```
> str_c(c("2021", "Feb", "15"), collapse = "-")
```

```
[1] "2021-Feb-15"
```

Combining strings

- The **str_c( )** function can do vectorized combination of two or more strings.
- Similar to unite( ) function

```
> covid %>% mutate(
  date = str_c(year, month, day, sep = "-")
)
```

```
# A tibble: 3 x 5
  country     year  month day    cases
  <chr>       <chr> <chr> <chr>  <dbl>
1 China        2021  01  22        98886
2 Japan        2021  01  22       357174
3 Philippines 2021  01  22       505939
```

```
# A tibble: 3 x 6
  country      year  month day    cases      date
  <chr>       <chr> <chr> <chr>  <dbl>      <chr>
1 China        2021  01  22       98886 2021-01-22
2 Japan        2021  01  22      357174 2021-01-22
3 Philippines 2021  01  22      505939 2021-01-22
```

Subsetting strings

- Use the **str_sub( )** function to extract parts of a string.
- The **start** and **end** arguments will indicate the positions of the substring.

```
> string_1 <− "This sentence is a string."
> str_sub(string_1, start = 1, end = 4)
```

```
[1] "This"
```

- Negative values count backwards from the end

```
> str_sub(string_1, start = -7, end = -2)
```

```
[1] "string"
```

Upper and lower cases

- Use the **str_to_upper( )** function to change the text to **uppercase**.
- Use the **str_to_lower( )** function to change the text to **lowercase**.

```
> string_1 <−  "This sentence is a string."
> str_to_upper(string_1)
```

```
[1] "THIS SENTENCE IS A STRING."
```

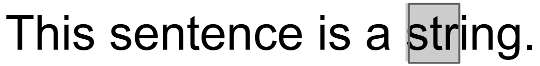
```
> string_1 <−  "This sentence is a string."
> str_to_lower(string_1)
```

```
[1] "this sentence is a string."
```

Regular expressions (REGEX)

- REGEX allow you to describe patterns in strings.
- Use the **str_view( )** function to see strings that matched the pattern.
- Take a character vector and a regular expression as inputs.

```
> string_1 <−  "This sentence is a string."
> str_view(string_1, "str")
```

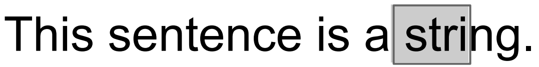


```
This sentence is a string.
```

Basic matches

- Use the period **“.”** to match any character except new line.

```
> string_1 <−  "This sentence is a string."
> str_view(string_1, ".str.")
```

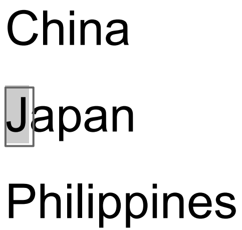


This sentence is a string.

Anchors

- Use **anchors** in the regular expression to specify the start or end of the string.
- caret **^** is used to match the **start** of the string

```
> country <- c("China", "Japan", "Philippines")
> str_view(country, "^J")
```



```
China

Japan

Philippines
```

Anchors

- Use **anchors** in the regular expression to specify the start or end of the string.
- dollar sign **$** is used to match the **end** of the string

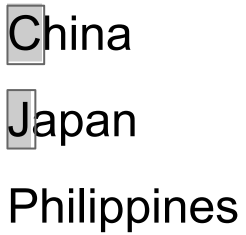
```
> country <- c("China", "Japan", "Philippines")
> str_view(country, "a$")
```

```
China

Japan

Philippines
```

Character classes

- Special patterns that match more than one character:
  \d for matching any digit
  \s for matching any whitespace (e.g. space, tab, newline)
  [abc] for matching a, b, or c
  [^abc] matches anything except a, b, or c

```
> country <- c("China", "Japan", "Philippines")
> str_view(country, "[CJ]")
```



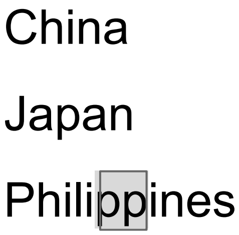
```
China

Japan

Philippines
```

Repetition

- Regex for controlling the number of times a pattern matches:

?	0 or 1
+	1 or more
*****	0 or more

```
> country <- c("China", "Japan", "Philippines")
> str_view(country, "pp+")
```



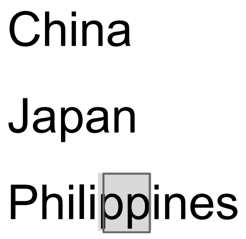
```
China

Japan

Philippines
```

Quantifiers

- Regex for controlling **exactly** the number of times a pattern matches:

{n}	exactly n
{n,}	n or more
{,m}	at most m
{n,m}	between n and m

```
> country <- c("China", "Japan", "Philippines")
> str_view(country, "pp{1}")
```

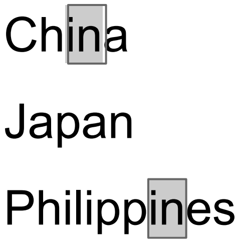


```
China

Japan

Philippines
```

Grouping

- Grouping allows parsing of values within the defined group.
- Parentheses are used to create groups.
- A capturing group stores the part of the string matched by the part of the regular expression inside the parentheses.

```
> country <- c("China", "Japan", "Philippines")
> str_view(country, "(in)")
```



```
China
Japan
Philippines
```

stringr tools

- Determine which strings match a pattern
- Find the positions of matches
- Extract the content of matches
- Replace matches with new values
- Split a string based on a match

Detect matches

- Use the **str_detect( )** function to determine if a character vector matches a pattern.
- It returns a logical vector the same length as the input.

```
> countries <- c("Afghanistan","Bangladesh", "China", "Japan", "Philippines")
> str_detect(countries, "[Aa]")
```

```
[1]  TRUE  TRUE  TRUE  TRUE FALSE
```

```
# How many countries end with letter “n”?
> sum(str_detect(countries, "n$"))
```

```
[1]  2
```

Detect matches

- Use the **str_count( )** function to determine the number of matches

```
> countries <- c("Afghanistan","Bangladesh", "China", "Japan", "Philippines")
> str_count(countries, "n")
```

```
[1] 2 1 1 1 1
```

Extract matches

- Use the **str_extract( )** function to extract the actual text of a match.
- It returns a character vector for the pattern or NA if match is not found.

```
> countries <- c("Afghanistan","bangladesh", "China", "japan", "Philippines")
> str_extract(countries, "[A-Z]")
```

```
 [1] "A" NA  "C" NA  "P"
```

```
> str_extract(countries, "[A-Z][a-z]+")
```

```
[1] "Afghanistan" NA          "China"        NA          "Philippines"
```

Extract matches

- Use the **str_extract_all( )** function to parse all matches and return a matrix of character vectors.

```
> str_extract_all("3y2m", "\\d")
```

```
[[1]]
[1] "3" "2"
```

Replacing matches

- Use the **str_replace( )** function to a replace matches with new strings.

```
> countries <- c("Afghanistan","bangladesh", "China", "japan", "Philippines")
> str_replace(countries, "^[a-z]+", "X")
```

```
[1] "Afghanistan" "X"           "China"       "X"           "Philippines"
```

```
> str_replace("Country/Region", "/", "_")
```

```
[1] "Country_Region"
```

Splitting

- Use **str_split( )** function to split a string up into pieces.

```
> string_1 <− "This sentence is a string."
> str_split(string_1, " ")
```

```
[1] "This"      "sentence" "is"     "a"         "string."
```

Find matches

- Use **str_locate( )** and **str_locate_all( )** functions to return the starting and ending positions of each match

```
> countries <- c("Afghanistan","Bangladesh", "China", "Japan", "Philippines")
> str_locate(countries, "a")
```

```
      start end
[1,]  5   5
[2,]  2   2
[3,]  5   5
[4,]  2   2
[5,]  NA  NA
```

Find matches

```
> countries <- c("Afghanistan","Bangladesh", "China", "Japan", "Philippines")
> str_locate_all(countries, "a")
```

```
[[1]]
      start end
[1,]  5   5
[2,]  10  10

[[2]]
      start end
[1,]  2   2
[2,]  6   6

[[3]]
      start end
[1,]  5   5

[[4]]
      start end
[1,]  2   2
[2,]  4   4

[[5]]
      start end
```

Take-away message

- **stringr** functions are useful in handling strings
- manipulating strings is challenging
- trial and error process