Lecture 17 - Regular Expressions

Packages

```
# none! we will use base R
```

String Processing with Regular Expressions

- Last lecture, we learned how to process and manipulate strings in `R` using base `R` functions
- In particular, we learned a set of functions that could search for a ***fixed*** string pattern in a character vector
  - `grepl(pattern, x)`
    - Outputs a logical representing if a `pattern` is in a string `x`
  - `grep(pattern, x)`
    - Outputs which elements in a character vector `x` contains a `pattern`
  - `gsub(pattern, replacement, x)`
    - Replaces a `pattern` in string `x` with another string `replacement`
  - `gregexpr(pattern, text)`
    - Outputs the location of the first character of a `pattern` in a string `text`

- `grepl(pattern, x)`
  - Outputs a logical representing if a `pattern` is in a string `x`

```
# define strings
strings <- c("Panda Express", "Vallarta's")

# search for "Express" in a character vector
grepl(pattern = "Express",
      x       = strings)
```

```
TRUE · FALSE
```

- `grep(pattern, x)`
  - Outputs which elements in a character vector `x` contains a `pattern`

```
# define strings
strings <- c("Panda Express", "Vallarta's")

# search for "Express" in a character vector
grep(pattern = "Express",
     x       = strings)
```

```
1
```

```
# Extract all strings with the pattern
grep(pattern = "Express",
     x       = strings,
     value   = TRUE)
```

```
'Panda Express'
```

- `gsub(pattern, replacement, x)`
  - Replaces a `pattern` in string `x` with another string `replacement`

```
# define strings
strings <- c("Panda Express", "Vallarta's")

# search for "Express" in a character vector and replace with "express"
gsub(pattern     = "Express",
     replacement = "express",
     x           = strings)
```

'Panda express' · 'Vallarta\'s'

- `gregexpr(pattern, text)`
  - Outputs the location of the first character of a `pattern` in a string `text`

```
# define strings
strings <- c("Panda Express", "Vallarta's")

# Show the location of "Express" in a character vector
gregexpr(pattern = "Express",
         text    = strings)
```

1. 7
2. -1

- But what if the patterns for which we are searching are ***NOT*** fixed?
- What if we want to search for a range of patterns, not just one string?

- For example, the following script uses the `grepl()` function to determine if a string in a character vector contains the pattern `"4"`
- The second entry of the logical vector output is `TRUE`, implying the second string in `vec` contains the pattern `"4"`

```
# define vector
vec <- c(    "My height is 5'6\"",
             "5'11\"",
             "I'm 6'3\"",
             "5'2\"",
             "5'4\" tall",
             "HEIGHT 4'10\""    )

# use grepl() to determine which entries contain the pattern "4"
result <- grepl("4", vec)
```

```
# print as new lines
cat(result, sep = "\n")
```

```
     FALSE
     FALSE
     FALSE
     FALSE
     TRUE
     TRUE
```

- But what if we want to search for a range of patterns (e.g., a range of numbers from 2-4)
- This can be performed using an approach called regular expressions
- Regular expressions use sequences of characters to search for a variety of patterns within strings

```
# define vector
vec <- c(    "My height is 5'6\"",
             "5'11\"",
             "I'm 6'3\"",
             "5'2\"",
             "5'4\" tall",
             "HEIGHT 4'10\""    )

# use grepl() to determine which entries contain a digit 2, 3, or 4
result <- grepl("[2-4]", vec)

# print as new lines
cat(result, sep = "\n")
```

```
     FALSE
     FALSE
     TRUE
     TRUE
     TRUE
     TRUE
```

Matching a range of numbers

- The script below searches for any occurrence of the characters `"2"`, `"3"`, or `"4"` in the character vector
- Single digit ranges are specified with square brackets and a hyphen `"[2-4]"`
- We read the script as follows:
  - Search for a string containing a single numeric digit between 2 and 4

```
# define vector
vec <- c(    "My height is 5'6\"",
             "5'11\"",
             "I'm 6'3\"",
             "5'2\"",
             "5'4\" tall",
             "HEIGHT 4'10\""    )

# use grepl() to determine which entries contain a digit 2, 3, or 4
result <- grepl("[2-4]", vec)

# print as new lines
cat(result, sep = "\n")
```

```
FALSE
FALSE
TRUE
TRUE
TRUE
TRUE
```

- To view which entries in the character vector satisfy your regular expression pattern, there are two options
  - `grepl()` outputs a `TRUE` if a string entry contains the pattern. We can use these logicals to subset the original character vector
  - An alternative is to use `grep()` with the `value = TRUE` option

```
# use grepl() to determine which entries contain a digit 2, 3, or 4
result <- grepl("[2-4]", vec)

cat(vec[result], sep = "\n")
```

```
I'm 6'3"
5'2"
5'4" tall
HEIGHT 4'10"
```

```
# use grep() to determine which entries contain a digit 2, 3, or 4
```

```
result <- grep("[2-4]", vec, value = TRUE)

cat(result, sep = "\n")
```

```
I'm 6'3"
5'2"
5'4" tall
HEIGHT 4'10"
```

Matching a range of letters

- Similar to numbers, we can also match a range of letters, which are specified with square brackets and a hyphen `"[a-z]"`
- The script below searches for any occurrence of a letter from lower case `"a"` to `"z"`
- We read the script as follows:
  - Search for a string containing a letter from `"a"` to `"z"`
- Notice the last entry was not matched because the letter is in upper case

```
# define vector
vec <- c(    "My height is 5'6\"",
             "5'11\"",
             "I'm 6'3\"",
             "5'2\"",
             "5'4\" tall",
             "HEIGHT 4'10\""    )

# use grepl() to determine which entries contain a lower case letter in the alphabet
result <- grepl("[a-z]", vec)

# print as new lines
cat(result, sep = "\n")
```

```
TRUE
FALSE
TRUE
FALSE
TRUE
FALSE
```

- We read the script as follows:
  - Search for a string containing a letter from `"a"` to `"i"`

```
# define vector
vec <- c(    "My height is 5'6\"",
             "5'11\"",
             "I'm 6'3\"",
             "5'2\"",
             "5'4\" tall",
             "HEIGHT 4'10\""    )

# use grepl() to determine which entries contain the letter a through i
result <- grepl("[a-i]", vec)

# print as new lines
cat(result, sep = "\n")
```

```
TRUE
FALSE
FALSE
FALSE
TRUE
FALSE
```

- You can also search for a range of upper case letters
- We read the script as follows:
  - Search for a string containing a letter from `"A"` to `"I"`

```
# define vector
vec <- c(    "My height is 5'6\"",
             "5'11\"",
             "I'm 6'3\"",
             "5'2\"",
             "5'4\" tall",
             "HEIGHT 4'10\""    )

# use grepl() to determine which entries contain the letter A through I
result <- grepl("[A-I]", vec)

# print as new lines
cat(result, sep = "\n")
```

```
FALSE
FALSE
TRUE
FALSE
```

```
FALSE
TRUE
```

- You can also mix the two together
- We read the script as follows:
    - Search for a string containing a letter from `"a"` to `"i"` or `"A"` to `"I"` or

```
# define vector
vec <- c(    "My height is 5'6\"",
             "5'11\"",
             "I'm 6'3\"",
             "5'2\"",
             "5'4\" tall",
             "HEIGHT 4'10\""    )

# use grepl() to determine which entries contain the letter a to i or A to I
result <- grepl("[a-iA-I]", vec)

# print as new lines
cat(result, sep = "\n")
```

```
TRUE
FALSE
TRUE
FALSE
TRUE
TRUE
```

- You can also separate letters by a comma
- We read the script as follows:
    - Search for a string containing a letter either the letter `"H"` or `"h"`

```
# define vector
vec <- c(    "My height is 5'6\"",
             "5'11\"",
             "I'm 6'3\"",
             "5'2\"",
             "5'4\" tall",
             "HEIGHT 4'10\""    )

# use grepl() to determine which entries contain the letter h or H
result <- grepl("[H,h]", vec)
```

```
# print as new lines
cat(result, sep = "\n")
```

```
     TRUE
     FALSE
     FALSE
     FALSE
     FALSE
     TRUE
```

OR Operator `"|"`

OR `"|"`

- In regular expressions, there is also an OR operation using the vertical bar `"|"`
- We read the script as follows:
  - Search for a string containing a letter from `"a"` to `"i"` OR a letter from `"A"` to `"I"`

```
# define vector
vec <- c(    "My height is 5'6\"",
             "5'11\"",
             "I'm 6'3\"",
             "5'2\"",
             "5'4\" tall",
             "HEIGHT 4'10\""    )

# use grepl() to determine which entries contain the letter a to i OR A to I
result <- grepl("[a-i]|[A-I]", vec)

# print as new lines
cat(result, sep = "\n")
```

```
     TRUE
     FALSE
     TRUE
     FALSE
     TRUE
     TRUE
```

Matching the Start (`"^"`) or End of a String (`"$"`)

`"^"`

- The start of a string is matched using a caret symbol `"^"`
- We read the following script as follows:
  - Search for a string ***starting*** with a letter from `"a"` to `"i"` OR ***starting*** with a letter from `"A"` to `"I"`

```
# define vector
vec <- c(    "I love it! Panda Express that is...",
             "i love Panda Express",
             "Panda Express is great!"    )

# search for strings starting with a through i or A through I
result <- grepl("^[a-i]|^[A-I]", vec)

# print as new lines
cat(result, sep = "\n")
```

```
TRUE
TRUE
FALSE
```

- If removing the starting caret symbol `"^"` from the first entry, notice how all three strings produce `TRUE`
- This is because the `"i"` in `"is"` is matched despite not starting the string
- We read the following script as follows:
  - Search for a string ***containing*** a letter from `"a"` to `"i"` OR ***containing*** a letter from `"A"` to `"I"`

```
# define vector
vec <- c(    "I love it! Panda Express that is...",
             "i love Panda Express",
             "Panda Express is great!"    )
```

```
# search for strings with a through i or starting with A through I
result <- grepl("[a-i]|^[A-I]", vec)

# print as new lines
cat(result, sep = "\n")
```

```
     TRUE
     TRUE
     TRUE
```

"$"

- The end of a string is matched using a dollar symbol `"$"`
- We read the following script as follows:
    - Search for a string ***ending*** with an exclamation point `"!"`

```
# define vector
vec <- c(    "I love it! Panda Express that is...",
             "i love Panda Express",
             "Panda Express is great!"    )

# use grepl() to determine which entries END WITH an exclamation point "!"
result <- grepl("!$", vec)

# print as new lines
cat(result, sep = "\n")
```

```
     FALSE
     FALSE
     TRUE
```

- Removing the `"$"` then matches the first and third string
- We read the following script as follows:
    - Search for a string ***containing*** an exclamation point `"!"`

```
# define vector
vec <- c(    "I love it! Panda Express that is...",
             "i love Panda Express",
             "Panda Express is great!"    )

# use grepl() to determine which entries contain an exclamation point "!"
result <- grepl("!", vec)
```

```
# print as new lines
cat(result, sep = "\n")
```

```
TRUE
FALSE
TRUE
```

Special Characters - `"."`, `"*"`, `"?"`, `"+"`

`"*"`

- The asterisk `"*"` represents ***zero or more*** instances of the previous character
- For example, we read the following script as follows:
  - Search for a string ***containing*** an `"A"`, then a `"1"` zero or more times, then a `"B"`
- All strings return a `TRUE` since all strings contain an `"A"`, zero or more instances of `"1"`, then one `"B"`

```
# define vector
vec <- c(    "AB",
             "A1B",
             "A11B",
             "A111B",
             "A1111B")

# grepl search with "*"
result <- grepl("A1*B", vec)

# print as new lines
cat(result, sep = "\n")
```

```
TRUE
TRUE
TRUE
TRUE
TRUE
```

"+"

- The plus symbol `"+"` represents ***one or more instances*** of the previous character
- For example, we read the following script as follows:
  - Search for a string ***containing*** an `"A"`, then a `"1"` ***one or more*** times, then a `"B"`
- The first string returns a `FALSE` since `"1"` does not occur one or more times
- The remaining strings return a `TRUE` since these strings contain a `"1"` at least one time

```
# define vector
vec <- c(    "AB",
             "A1B",
             "A11B",
             "A111B",
             "A1111B")

# grepl search with "+"
result <- grepl("A1+B", vec)

# print as new lines
cat(result, sep = "\n")
```

```
FALSE
TRUE
TRUE
TRUE
TRUE
```

"?"

- The period `"?"` represents ***none or one*** instance of the previous character.
- For example, we read the following script as follows:
  - Search for a string ***containing*** an `"A"`, then `"1"` ***zero or one*** time, then a `"B"`
- The first two strings return a `TRUE` since the character `"1"` occurs ***zero or one*** time

```
# define vector
vec <- c(    "AB",
             "A1B",
             "A11B",
             "A111B",
             "A1111B")

# grepl search with "?"
```

```
result <- grepl("A1?B", vec)

# print as new lines
cat(result, sep = "\n")
```

```
TRUE
TRUE
FALSE
FALSE
FALSE
```

"."

- The period `"."` represents ***any single*** character
- For example, we read the following script as follows:
  - Search for a string containing an `"A"`, then any single character, then a `"B"`
- Notice how only the second string `"A1B"` results in a `TRUE`
- All other strings have zero or more than one character between `"A"` and `"B"`, resulting in `FALSE`

```
# define vector
vec <- c(    "AB",
             "A1B",
             "A11B",
             "A111B",
             "A1111B")

# grepl search with "."
result <- grepl("A.B", vec)

# print as new lines
cat(result, sep = "\n")
```

```
FALSE
TRUE
FALSE
FALSE
FALSE
```

Escaping

- Last lecture, we learned that we can use the backslash `\` to include specific formatting in a string
- For example, we require a single backslash `\` to include a quote in a string

```
# print a string with quotes
cat("I am 5'6\" tall")
```

```
     I am 5'6" tall
```

```
# without the escape \ outputs an error
cat("I am 5'6" tall")
```

```
     Error in parse(text = input): <text>:2:16: unexpected symbol
     1: # without the escape \ outputs an error
     2: cat("I am 5'6" tall
                       ^
     Traceback:
```

Étapes suivantes : Expliquer l'erreur

- In regular expressions, there are other characters that have a special meaning
- Escaping these special characters requires two backslashes `"\\"`

The Digit Special Character `"\\d"`

- The special character `"\\d"` is equivalent to specifying `["0-9"]`

```
# define vector
vec <- c(    "My height is 5'6\"",
             "5'11\"",
             "I'm 6'3\"",
             "5'2\"",
             "5'4\" tall",
             "HEIGHT 4'10\""    )

# search for a numerical digit
```

```
result <- grepl("[0-9]", vec)

# print as new lines
cat(result, sep = "\n")
```

```
     TRUE
     TRUE
     TRUE
     TRUE
     TRUE
     TRUE
```

```
# define vector
vec <- c(    "My height is 5'6\"",
             "5'11\"",
             "I'm 6'3\"",
             "5'2\"",
             "5'4\" tall",
             "HEIGHT 4'10\""    )

# use grepl() to determine which entries contain a numerical digit
result <- grepl("\\d", vec)

# print as new lines
cat(result, sep = "\n")
```

```
     TRUE
     TRUE
     TRUE
     TRUE
     TRUE
     TRUE
```

- You can specify the number of digits for which you are searching using curly braces `"\\d{number digits}"`
- For example, we read the following script as follows:
    - Search for a string containing 3 or 4 digits in a row

```
# define vector
vec <- c(    "My height is 5'6\"",
             "5'11\"",
             "I'm 6'3\"",
             "5'2\"",
             "5'4\" tall",
             "HEIGHT 4'1000\""    )
```

```
# use grepl() to determine which entries contain 3 or 4 digit numbers
result <- grepl("\\d{3,4}", vec)

# print as new lines
cat(result, sep = "\n")
```

```
FALSE
FALSE
FALSE
FALSE
FALSE
TRUE
```

The White Space Special Character `"\\s"`

- Often times, strings will contain white space
- It can be useful to search for white space to debug or correct strings for an analysis

```
# define vector
vec <- c(    "My height is 5'6\"",
             "5'11\"",
             "I'm 6'3\"",
             "5'2\"",
             "5'4\" tall",
             "HEIGHT 4'10\""    )

# use grepl() to determine which entries contain white space
result <- grepl("\\s", vec)

# print as new lines
cat(result, sep = "\n")
```

```
TRUE
FALSE
TRUE
FALSE
TRUE
TRUE
```

The [illegible] Function

The `regmatches()` Function

- Thus far, we have learned to find a range of patterns within a vector of strings
- But what if we want to extract the content of these patterns?
- For example, we may want to extract only heights and no other text

- We can use the `regmatches(x, m)` function!
  - `x`: a vector of character data types (strings)
  - `m`: output from the `gregexpr()` function

- The script below first uses the `gregexpr()` function to find the location of the start of a pattern
- Since we want to extract the heights, the pattern for which we are search reads as follows:
  - Search for patterns that start with a number `"\\d"`
  - followed by an single quote `"'"`
  - followed by a number with 1 or two digits `\\d{1, 2}`
  - followed by a double quote `"\""`

```
# define vector
vec <- c(    "My height is 5'6\", is that short?",
             "5'11\" maybe?",
             "I'm 6'3\"",
             "I am 5'2\" tall",
             "5'4\" tall",
             "HEIGHT 4'10\""    )

# find location of pattern
pattern_locations <- gregexpr("\\d\'\\d{1,2}\"", vec)

pattern_locations
```

1. 14
2. 1
3. 5
4. 6
5. 1
6. 8

- We can then use the `regmatches()` function and the pattern locations to extract the patterns themselves

```
# extract patterns from strings using locations
extracted_patterns <- regmatches(vec, pattern_locations)

# print patterns
cat(do.call(c,extracted_patterns), sep = "\n")
```

```
5'6"
5'11"
6'3"
5'2"
5'4"
4'10"
```

- Why is this useful?
  - This process provides us with the ability to extract data from text for analysis!

Examples

- Below are several examples of how you can use regular expressions to explore text data
- We will explore strings from a twitter (now X) dataset related to the covid-19 pandemic

```
# load data
tweets <- read.csv("https://raw.githubusercontent.com/khasenst/datasets_teaching/refs/

# show first few lines
head(tweets)
```

A data.frame: 6 × 4

user_name	user_location	date	text
<chr>	<chr>	<chr>	<chr>
		2020-07-25	If I smelled the scent of hand sanitizers today on someone in the past, I would think

1	ℽ°i☻⌊₵↑	astroworld	2020-07-25 12:27:21	today on someone in the past, I would think they were so intoxicated that… https://t.co/QZvYbrOgb0
2	Tom Basile 🇺🇸	New York, NY	2020-07-25 12:27:17	Hey @Yankees @YankeesPR and @MLB - wouldn't it have made more sense to have the players pay their respects to the A… https://t.co/1QvW0zgyPu
3	Time4fisticuffs	Pewee Valley, KY	2020-07-25 12:27:14	@diane3443 @wdunlap @realDonaldTrump Trump never once claimed #COVID19 was a hoax. We all claim that this effort to… https://t.co/Jkk8vHWHb3
		Stuck in the	2020-07-25	@brookbanktv The one gift #COVID19 has give me is an appreciation for the simple

```
# show lines of text column
head(tweets$text)
```

'If I smelled the scent of hand sanitizers today on someone in the past, I would think they were so intoxicated that… https://t.co/QZvYbrOgb0' · 'Hey @Yankees @YankeesPR and @MLB - wouldn\'t it have made more sense to have the players pay their respects to the A… https://t.co/1QvW0zgyPu' · '@diane3443 @wdunlap @realDonaldTrump Trump never once claimed #COVID19 was a hoax. We all claim that this effort to… https://t.co/Jkk8vHWHb3' · '@brookbanktv The one gift #COVID19 has give me is an appreciation for the simple things that were

- The covid-19 pandemic caused quite a bit of panic
- Let's see the pattern `"anxiety"` is included in the tweets

```
results <- grep("anxiety", tweets$text, value = TRUE)

print(head(results, 10))
```

```
 [1] "For me, someone who has been affected by social anxiety, wearing a face mask
 [2] "“The anxiety surrounding COVID-19 has caused Bipolar and Chron’s Disease to
 [3] "For the last few months I’ve used #running &amp; #singing as ways of reducin
 [4] "So happy #MLB baseball is back, but this 60 game season is causing major fan
 [5] "My new article on mask anxiety. Please share to help those struggling with a
 [6] "The Good Things From This Difficult Time... https://t.co/aeDljPsFpo | #covid
 [7] "Stay #Fit and #Healthy through regular #Meditation when you are at home to d
 [8] "How COVID Has Affected Me... https://t.co/QIRTVxMpyA | #covid19 #anxietyisre
 [9] "Battling anxiety and insecurity during #Covid times https://t.co/EURQCzUQ1b
[10] "Relaxing of strict lockdown rules can actually add to anxiety and stress. He
```

- What about tweets starting with the word `"Anxiety"`

```
# search for patterns starting with Anxi
results <- grep("^Anxi", tweets$text, value = TRUE)

print(head(results))
```

```
     [1] "Anxious days ahead for @AirAsia and @MAS, as second wave fears rise in #Malay
     [2] "Anxiety is up - women are more likely to feel overwhelmed (39% for women; 26%
```

- What about tweets starting with the word `"Depress"`

```
# search for patterns starting with Depress
results <- grep("^Depress", tweets$text, value = TRUE)

print(head(results))
```

```
     [1] "Depression and other mental health disorders can cause a weakened immune resp
     [2] "Depressing. Keep men off the streets, (non-marital) rape and sexual assault d
```

- What about tweets starting with a number?

```
# search for patterns starting with a number
results <- grep("^\\d", tweets$text, value = TRUE)

print(head(results))
```

```
     [1] "25 July : Media Bulletin on Novel #CoronaVirusUpdates #COVID19 \n@kansalrohit
     [2] "49K+ Covid19 cases still no response from \n@cbseindia29 @HRDMinistry @DrRPNi
     [3] "1 municipality reports one new fatality due to #COVID19, compared to yesterda
     [4] "2,000 women lawyers write to #AmitShah seeking 5 Lakh Loan per financially dr
     [5] "1.28% of the U.S. population is infected with Covid-19 \n#COVID19 \n#TrumpVir
     [6] "7813 new positive #COVID19 cases reported in #AndhraPradesh from last 24hours
```

- What about tweets starting with a number and including strings related to "depress"?
- We read the script as follows:
    - Search for a string starting with a number and containing "depress" OR "Depress"

```
# search for patterns starting and containing "depress" or "depress"
results <- grep("^\\d.*depress|^\\d.*Depress", tweets$text, value = TRUE)
```

```
print(head(results))
```

```
     [1] "10-15% of elderly suffer depression. Many are at risk of suicide. Another rea
```

- It seems like tweets are reporting some statistics
- Let's see of we can explore more of these reported numbers
- We read the script as follows:
    - Search for a string
        - starting with 1 or more numbers, then a comma OR
        - starting with 1 or more numbers then a decimal point OR
        - starting with 1 or more numbers then a percentage

What if we are looking for numbers people are reporting?

```
# search for strings starting with a number
results <- grep("^\\d+,|^\\d+\\.|^\\d+%", tweets$text, value = TRUE)

print(head(results, 12))
```

```
      [1] "2,000 women lawyers write to #AmitShah seeking 5 Lakh Loan per financially d
      [2] "1.28% of the U.S. population is infected with Covid-19 \n#COVID19 \n#TrumpVi
      [3] "1,142 #COVID19 positive cases, 2,137 patients recovered/discharged/migrated
      [4] "100% feeling good and 0% some symptoms😳 Long May it last 🤞 #COVID19 🦠🤛
      [5] "100%  feeling good ! Sounds like a soul song ! #COVID19 https://t.co/E5fp3sI
      [6] "1,005 new cases and 18 new deaths in the United States \n\n[13:16 GMT] #coro
      [7] "1,142 #COVID19 positive cases, 2,137 patients recovered/discharged/migrated
      [8] "10,000+ #healthworkers have been infected with #COVID19.@WHO's Dr Talisuna o
      [9] "5,144 new cases and 58 new deaths in India \n\n[13:05 GMT] #coronavirus #Cor
     [10] "5,144 new cases and 58 new deaths in India \n\n[12:58 GMT] #coronavirus #Cor
     [11] "144,000+ dead in the US from #COVID19 and not slowing down.  Thanks @GOP"
     [12] "173.19 MT Fresh Ginger has been exported from Manu LCS (#Tripura) to #Bangla
```

- What if we wanted to extract the relevant links?
- Notice that all of the links at the end of the tweet start with `"http"`
- We can use `gregexpr()` to determine the location of the link in the string

```
# find location of the "h" in the url
result_links <- gregexpr("http.*", results)

head(result_links)
```

1. 115
2. -1
3. 118
4. 74
5. 57
6. -1

- We can then use a function called `regmatches()` to extract those links from the `gregexpr()` output

```
# extract links from the tweets using gregexpr() output
links <- regmatches(results, gregexpr("http.*", results))

head(links)
```

1. 'https://t.co/JEjVTeCjEn'
2.
3. 'https://t.co/qF6QYCjcmd'
4. 'https://t.co/37Dc8BweJO'
5. 'https://t.co/E5fp3sIBl5'
6.

Commencez à coder ou à générer avec l'IA.

Hi class. Welcome to today's lecture. So today, we're going to cover more string processing. And in particular, we're going to cover something called regular expression.

Now, in last lecture we actually learned how to process and manipulate strings using several different base R functions, such as nchar for the number of characters in a string, toupper to convert strings to all uppercase, and substring to subset different patterns within a string.

But in particular, toward the end we also learned another set of functions that can search for a fixed string pattern in a vector of character data types — or a character vector, or you could call that a vector of strings.

So there are these functions here, very special regular expression functions. Now, just to recap: we have grepl. The "l" in grepl stands for logical, right? Because it outputs a Boolean or logical, which represents if a pattern is in a string.

So just to recap: if I defined this character vector here with two entries, the first being "Panda Express" and the second being "Vallartas," which are two different places to eat.

I can call grepl, and I want to say I'm searching for the pattern "Express" with a capital E in

each of the strings within this vector.

And if I run that, I get a TRUE and a FALSE. The first element is TRUE because the first string in this vector contains this pattern.

However, the second string in the vector does not.

So that's how grepl works.

There's also grep, which is very similar to grepl, but instead of outputting a logical vector, it outputs just a numeric vector — right — about which indexes or elements of the character vector contain that pattern.

So when I run it on this same character vector here, it outputs a single 1. And this is because the first entry of this character vector contains the pattern "Express."

It doesn't output anything for 2, because the second string in that character vector does not contain this pattern.

Now, something I didn't go over is: there is another option or argument for the grep function — it's called value. And the function of this argument value just simply outputs the corresponding string that contains your pattern.

So what I mean by that is: since the first element of this character vector contains "Express," if you put value = TRUE, it just gives you that entire string.

If I run that, it outputs the first entry of that character vector. This is useful when you don't want to just extract the numeric index or location of the string containing the pattern — it essentially outputs the entire string itself.

Then there's gsub, which just replaces a pattern in a string x with some replacement string. So again, if we're working with this same character vector — if I wanted to convert this "E" to a lowercase, I could say: search for this pattern in this character vector, and wherever you find that pattern, replace that pattern with this pattern here, which is just lowercase "e".

So if I show that, you can see now that there is indeed a lowercase "e" there.

And then finally, a bit more complicated is the gregexpr function. And this actually searches for the location of the pattern and outputs that location.

So if I were to say, "search in this character vector for this pattern and give me the location of the first character in that pattern," well, here we have a 7, and if we count:

There's 1, 2, 3, 4, 5, 6, and then the 7th character is the "E", so it gives us the location of this pattern in this longer string.

And for the second entry here, it outputted a -1, just because "Vallartas" doesn't contain anything related to this pattern here. So it outputs a negative number, saying there was no match. All right. So that's a quick recap.

Now, as I said earlier,

all of these functions — the way we learned them — was just searching for a fixed string pattern. So "Express" is just fixed — we are searching exactly for that pattern.

The thing is, what if these patterns that we're searching for aren't fixed? What if we want to search for a range of different patterns?

Here's an example:

Say we have this vector here of several descriptions of individual heights. The first is "my height is 5 foot 6", second one says "5 foot 11 inches", and so on.

Now, we could use grepl to search for this exact pattern of a "4" within this character vector,

and then print it.

You can ignore this, but this is just using the cat function and printing a new line in between each entry.

Notice here: it outputs a vector of logicals where the first four are FALSE and the last two are TRUE.

And the reason for this is: we have a "4" here and a "4" here in the last two entries, right? So there's a correct match there. However, there aren't any "4"s in the earlier entries.

The thing is, what if we want to search for a range of numbers,

such as all heights that contain a digit 2 to 4 — right — 2, 3, or 4? Well, you can perform this using the same function, but you need a different syntax called regular expressions.

So I'll just show a quick example here, but then I'll go in depth into how this works in future videos.

So I could say: grep — search for "2 to 4" — and then print the result. Now you see that the middle two are also TRUE.

So here we have a 4, we have a 4,

but in the middle two we have a "3", which is in the range 2 to 4,

and we have a "2", which is in the range 2 to 4 as well.

So we're actually able to search for a range of patterns — not only for numbers, but also letters — search for strings that start with certain letters, and so on, using this syntax of regular expressions.

Alright. So how did we find strings that contain that pattern of a range of numbers in the last lecture video? Right? Well, let's look at this script once again.

Single digit ranges in our pattern string that we're searching are specified with square brackets and a hyphen. So if you want to search for any digits in a string that are ranging from 2 to 4, you would include the quotes (because that's a string pattern), the bracket (which tells R "I'm going to include a range"), and then the numbers for which you'd like to search.

So again, this was the result from the last video.

Now, to view which entries in the character vector satisfy your regular expression pattern, there are two options. For one, remember grepl outputs a logical vector. So we can use subsetting to pull the entries in this character vector corresponding to TRUE.

If the last four are TRUE, we can use subsetting to pull the last four strings from that vector. That's what I do here.

So I have results, which is just this logical vector. I say I want from the vector everything that's TRUE.

And it simply outputs all the strings that satisfy the pattern of having a number ranging from 2 to 4. But again, if I used value = TRUE, like we went over earlier in the last video, we can simply print that result and it will print these strings as well.

Alright. So we've matched a range of numbers. Now we are working with text which will likely contain letters. So what if we want to match a range of letters?

We can do it using similar syntax from before — using that bracketed syntax. We just specify a range of letters by including that range of letters within square brackets and separating the range with a dash or a hyphen.

We read this script as: we would like to search for a string containing a lowercase letter, or anything from a lowercase letter a to a lowercase letter z. So this is the same character vector defined before.

Except now, I'm searching for anything that contains a letter from a to z. Remember, you need these square brackets and this hyphen, which signifies a range. If I run this and print the result, we get: TRUE, FALSE, TRUE, FALSE, TRUE, FALSE.

If we look at this result: the first is TRUE — the string contains at least one lowercase letter. The second string doesn't. The third does (an "m"). The fourth doesn't. The fifth does ("TALL"). Notice how the last doesn't, despite containing letters — because we're only searching for lowercase letters, whereas "HEIGHT" here is all uppercase.

Another example: we can read this script as follows — I want to search for a string that contains a lowercase letter a through i. So I'm narrowing the range to just a through i and searching through this character vector.

Notice we have TRUE, FALSE, and now the third one turns to FALSE. The reason is although "I" is contained in this third string, it's uppercase — so it's not included here. This is also uppercase, so it also outputs FALSE. No pattern match.

Just like searching for a range of lowercase letters, we can search for a range of uppercase letters using the same syntax, but with capital letters. So I would read this script as: search for a string in this character vector that contains a letter A through I.

If I run that, we get: FALSE, FALSE, and TRUE. Notice how this is now TRUE — that third entry

If I run that, we get: FALSE, FALSE, and TRUE. Notice how this is now TRUE — that third entry — because there's a capital I matching with the I here.

And the last entry is TRUE because we have an all-caps "HEIGHT" here.

All the other ones are FALSE, because we're not matching any lowercase letters, and the capital M is not within the range of A through I.

You can also mix the two together by including both lowercase and uppercase ranges in that same bracket syntax.

I would read this script as: search for a string in this character vector that contains a letter in the range of lowercase a through i and uppercase A through I.

Now you see: TRUE, FALSE, TRUE, FALSE, TRUE, TRUE, TRUE — basically anything with text is TRUE. The second and fourth entries are FALSE, because they do not contain any text at all.

One more thing with letters: you can also separate them by a comma if you don't want to search for a range, but just a few specific letters. Here I'm changing the pattern to Hh — uppercase and lowercase H.

We are searching for a string in this character vector containing an uppercase H or a lowercase h.

If I run that, you'll see that everything is FALSE except the first and last entries. The first contains a lowercase h, and the last contains an uppercase H.

So that's searching for ranges of letters or patterns containing letters.

WEBVTT

1 00:00:02.610 --> 00:00:13.310 Kyle: So in regular expressions. You can also have Boolean operators. In particular, there is the or operator, which is also represented by a vertical bar, just like when we were subsetting.

2 00:00:13.430 --> 00:00:42.669 Kyle: However, you need to include this vertical bar within the quotes because we are searching for patterns within strings. Okay, so here's a script which illustrates how to use that vertical bar. So we would read the script as follows, we're searching for a string containing a letter right? We're searching for a string within this character vector that contains a lowercase letter, a through I or that contains an uppercase letter, a through. I okay, so running that

3 00:00:42.790 --> 00:00:52.799 Kyle: we get a true, false, true, false, true, true. And this is because we're searching for anything containing any letter, lowercase, or uppercase a through. I

4 00:00:53.140 --> 00:01:07.430 Kyle: obviously you have false and false here. This is because these are just heights containing 0 letters, right? And this is essentially the same as the example above. If I scroll up here where you have a through, I or a through I.

5 00:01:07.640 --> 00:01:12.679 Kyle: But the the utility of that or operator will become more apparent later in lecture.

6 00:01:14.130 --> 00:01:16.430 Kyle: Okay? So this is the or operator.

Now, instead of matching a range of numbers or a range of letters occurring anywhere in a string within a character vector, we can actually match letters or numbers at the start or end of a string as well. Here's an example of using regular expressions to find a pattern at the start of a string.

The start of a string is matched using the caret symbol (^) — this little arrow pointing up. So the caret symbol is placed in quotes.

We would like to search for a string in this character vector that starts with the letter a through i, or starts with the letter A through I. The way we do this is: we want to start with any of the range of letters inside brackets, either lowercase or uppercase A through I.

When we run that, we only have three entries within this character vector — the first two produce TRUE.

This is because the first letter in the first entry starts with a capital I, which is in the specified range. The first letter in the second entry starts with a lowercase i, also in the range. However, the third entry starts with a capital P, which is not within the range of A through I.

Now, if we remove the starting caret symbol from that first entry, all three strings produce TRUE.

This is because the i in the middle is still matched. So the caret symbol (^) essentially allows you to search for patterns at the beginning of a string.

Similarly, you can use the dollar symbol ($) to search for patterns at the end of a string.

We would like to search for a string within this character vector that ends with an exclamation point.

In our grep statement, we're searching this character vector for an exclamation point at the end. We use the dollar symbol to indicate we're searching for a character at the end of the string. The character we're searching for comes before the dollar symbol.

So we want the string to end with an exclamation point.

When we run this, we get FALSE, FALSE, and TRUE, because the third string ends with an exclamation point.

Now, if we were to remove the dollar symbol and just search for an exclamation point, then the first and third strings return TRUE, because there's an exclamation point in the middle of the first string and at the end of the third.

Alright. So there are many, many special characters that we can use within regular

expressions to search for string patterns. I'll just cover a few here today — the most important ones.

First, we have the asterisk (*). This represents zero or more instances of a character that is shown previously. What that means is: if we look at this script, it reads — we are searching for a string in this character vector containing:

```
an "A",

then a "1",

but that "1" can occur zero or more times (because of the asterisk),

followed by "AB".
```

If we run this, all of them are TRUE.

Why? Because our rule is that we want it to start with an "A" — and all the strings in the character vector start with "A". Then we need a "1". All of these have a "1", except for the first one. But because of the asterisk, we're saying the "1" can occur zero or more times — so the first entry still qualifies. Then it's followed by "AB".

The "1" can occur zero or more times. For example, it occurs four times in one entry, followed by "AB", so there's still a match.

Now, similar to the asterisk, we can use a plus symbol (+).

The plus symbol represents one or more instances. So if we take the same example but replace the asterisk with a plus:

We are searching for a string that:

```
contains an "A",

then contains a "1" one or more times,

followed by "AB".
```

If we run this, the first string returns FALSE, and everything else is TRUE. That’s because the first string starts with an "A", but there are zero "1"s, and we need one or more. The rest have at least one "1", so they satisfy the pattern.

Now, the question mark (?) represents zero or one instance of the preceding character — not more than one.

So in the script using the question mark, we’re searching for a string that:

```
contains an "A",

then a "1" that occurs zero or one time,

followed by a "B".
```

If we run this, the first two strings return TRUE, and the rest are FALSE.

Why? All strings start with "A". But for the first two, "1" occurs zero or one time. For the others, it occurs two or more times, which does not satisfy the condition.

And finally, the period (.) represents any single character — it can be a number, letter, or symbol.

So if the pattern is:

```
starts with "A",

followed by any single character,

then "B",
```

Only the second string is TRUE.

Why? The first string starts with "A" and ends with "B", but there is no character in between — so it's not a match. Others have more than one character between "A" and "B". Only the second has exactly one character in between.

If we modify the first string, for example changing it to "A7B", now the pattern matches — "A" followed by a single character ("7") and then "B" — and the result is TRUE.

Alright. Now we're getting into more advanced regular expressions. Just like with defining strings in the last lecture, we can also use escape characters — or escaping. In the last lecture, we used a single backslash to include some specific formatting like a quote.

For example, to include this quote so it doesn't conflict with the surrounding quotes, we had to put a backslash to let R know we actually want to include that quote inside the string.

If we don't use the escape character, R gets confused — and we end up with an error.

In regular expressions, there are other special characters as well, and they have special meanings. But they can conflict with the single backslash in a regular string, so we end up needing two backslashes when using regular expressions in R.

The first such special character requiring escaping is the digit special character, written as \d.

This is equivalent to specifying a range of digits 0-9. So if we search in a character vector for

any pattern containing digits 0–9, we'll find matches in all strings containing numbers.

Instead of using [0-9], we can write \d — a shorthand for any single digit. You need to use two backslashes (\) or else you'll get an error, because \d is a special regex character and needs to be escaped in R strings.

With two backslashes, the results are equivalent to the earlier example — TRUE for each entry that contains at least one digit.

Now, you can also specify the number of digits you're searching for using curly braces {} after \d.

For example: \d{3,4} means you are searching for a string containing three or four digits in a row.

If you modify an entry to have four digits in a row and run this search, it will return TRUE for that entry.

You can try other combinations too. For instance, \d{2} will match only entries with exactly two digits in a row.

So if an entry has only one or more than two digits in a row, it won't match.

This is helpful when searching for numeric patterns with a specific number of digits, such as phone numbers, ZIP codes, etc.

A second special character we'll cover is the whitespace special character, written as \s.

Often, strings may contain inconsistent or excess whitespace, and you may want to search for or replace these. You might also want to detect words separated by spaces.

If you search using \s, it will return TRUE for strings that contain at least one whitespace character.

For example, if we run this on a character vector, you might see something like: TRUE, FALSE, TRUE, FALSE, TRUE, TRUE.

Here, the second and fourth entries return FALSE because they have no spaces.

However, other entries include spaces — like between "I'm" and "6 foot 3", or between "My height" and "is".

So, detecting whitespace with \s is useful when processing real-world text data, especially where formatting is messy or inconsistent.

Alright. So we're almost done covering all the new content before we jump into a sort of longer example. But first I want to cover a very useful and important function called regmatches. This is a new function that can be used in conjunction with some of the regular expression functions to extract the patterns for which we're searching — not the entire string, but the actual pattern.

So far, we've learned to find ranges of patterns within a vector of strings — ranges of numbers and letters. But if we want to extract that actual pattern we're matching — for example, we have a vector of height descriptions, and we want to extract just the height for data analysis — we can use regmatches.

This function takes two main arguments (there is a third optional one):

```
x, which is your character vector — the vector of strings we've been working with

m, which should be the output of a regular expression function that shows the locatio
```

Using the locations from m, regmatches extracts the actual matching text pattern.

For example, we can use gregexpr() to find the location of a pattern in a string. Here's what we're looking for: a number (e.g., "5"), followed by an apostrophe (foot symbol), followed by another number with one or two digits, followed by a quotation mark (inch symbol).

So in the pattern:

```
a number

an apostrophe '

one or two digits

a quote "
```

This would match something like 5'10".

If we run gregexpr() on a character vector of height descriptions, we get the positions in the strings where these patterns start.

For example, if the 14th character in the string is the start of the height pattern, it will return 14. For another entry, the pattern might start at the first character.

Now, instead of manually trying to extract that substring, we can pass this output into regmatches along with the original vector.

regmatches(x, m) will extract the exact substring that matched the regular expression — in this case, the height.

Running this will give you a list of the matched patterns. You can clean this up by using do.call() to concatenate the list into a flat vector, and print it using cat().

Now we have a clean list of height values extracted from messy text! This is really useful — if we wanted to build a dataset for analysis, we could scrape height values like this and insert them into a data frame.

All right, let's so let's go over some examples to see how this can be used in practice. Alright so we're gonna explore strings from an X which is now X right? Twitter data set, related to the COVID-19 pandemic. And it's available from this link from my Github.

So I'm going to load that data using, read dot Csv

and show you the head. So the 1st few lines. Now, you can see, we have the person's username, the location, the date of the post, as well as the text from that post.

Alright. Now we can just show you the text itself. Just so you could see that right? This is a character vector, so the text column

from the Tweets data frame is simply a character. Vector, so we can use all of the character or the string processing and regular expressions. Code to work with this.

Right? So say you have this data set, and you're just you don't even know what questions to ask. You're just thinking about, how can I analyze these data?

Right? Well, the COVID-19 pandemic

obviously, cause caused quite a bit of global panic. So maybe you want to search for you know, any posts related to the word anxiety.

So what we'll use is Grep, we're going to use regular expressions to search this Tweet character vector or the pattern anxiety. And notice, this is a fixed pattern. I'm not using any of the special syntax that we've covered today. Yet.

Alright. So if I print the 1st 10 results you could see here for me someone who has been affected by social anxiety wearing a face mask is amazing. So this person seems like they like the pandemic simply because they can wear a face mask.

There's another person who says the anxiety surrounding Covid-nineteen has caused Bipolar and Crohn's disease to flare up. That's very unfortunate. Those are 2 very, very difficult health problems to deal with, and so on. Right? So this is quite interesting. You can already start searching for and seeing what people are saying related to anxiety.

although this one's unrelated saying, Major Fantasy League anxiety. Okay.

now, what if we want to search for the capital word anxiety?

Moreover, what if we want to search for strings that start with just A and Xi for anxious or anxiety. We want to search for all of those

right.

So there are only 2 tweets in this data set.

One talks about anxious days ahead for Airasia.

Related to Covid. So they're talking about the flights and the fear surrounding that

and talking about anxieties up. Women are more likely to be overwhelmed. I'm not sure how

true that is this. Remember, these are tweets. So you need to take whatever facts, they say, quote unquote facts, they say, with a grain of salt. Right?

What if you want to look for depression? So you're looking at depressed any strings that start with depressed

right?

Something I didn't mention is, we are using Grep, not grep, L, because I want to actually use value equals. True, to extract the text itself. Okay?

So looking at this, we have 2 strings that matched

depression and other mental health disorders can cause a weakened, immune response, which is definitely true.

If it affects your immune system, you can get sick. You have depressing. Keep men off the streets. There's some

wildly crazy and inappropriate tweets going on here. Right?

What about tweets that start with a number? Say, we want to think about? I want to extract some data from these tweets, some relevant statistics that may or may not be true.

Right? So I'm going to say, search in this character vector for any

patterns or any strings with patterns that start right. The carrot start with a digit from 0 to 9 or this special character here.

and I'm saying, value equals true to print all the results

right? And you could see there are some statistics here. So 7,813 new positive cases. And already you can see how you can use tweets to extract some data, or even look at the the frequency of people mentioning certain words to correlate with some event. Right?

Alright. So we have something here, another 1 49,000 plus COVID-19 cases. Still no response.

Okay.

now, what about tweets starting with a number and including strings related to say depression, right? So I'm gonna write this whole pattern here.

It looks like a lot. But it's not too much right. It's saying, search for a string

that starts with right. The caret symbol starts with a digit from 0 to 9.

Then, after that digit, there's a period here. So any single character.

However, it's followed by an asterisk. So I'm saying, this digit is followed by any number of characters, 0 or more characters

then followed by the string to press.

Okay, so I'm searching for that entire pattern, or

I want it to start with a digit

and be followed by any character that occurs any number of times.

and then the pattern depressed with a capital D

alright. So I'm searching for anything that starts with a number, and then is followed by

depressed or contains the word depression.

Alright. So already we just have a single match. You can see here, it says 10 to 15%, because it starts with a number

of elderly suffer. Unfortunately, from depression, many are at risk of suicide. This just sounds true, right? But you'd obviously have to verify.

and we've matched it successfully. Okay.

now, it does seem like these tweets are reporting some statistics. So let's see if we can explore more of these numbers. Right?

So I'm going to write out this pattern here.

Right? So we're searching for a string in this character vector of tweets that starts with a number

that occurs one or more times.

Okay? And then may contain a comma. Right? So we're searching for numbers in the thousands or 100 thousands.

And then we have a vertical bar here denoting, or the or operator.

or we want to search for a string that starts with a number one or more times, and is followed by a period. So now we're searching for decimals.

Finally, or we're searching for strings that start with a number

with one or more numbers and then

has a percentage symbol. So searching for any percentages there.

Now, if I run this.

you could see that many tweets are reporting certain statistics, and you could start evaluating whether or not they're true or verifying them. This one's obviously not

relevant. It says 100% feeling good. So this person is just unaffected completely by the pandemic. Right now.

these are nice. We can extract those numbers like we did before. But what if we wanted to extract, say, the relevant links just to see the complete tweet sometimes. You don't have the complete tweet right? What if we wanted to extract the relevant links just to verify and, you know, go forward with our analysis.

Well, we can search for any strings that contain the pattern, Http, right?

Well, we can search for any strings that contain the pattern. Http, right?

Because all of these end with Http or https.

Okay, so what I'm going to do is use this Gregx function to search

only these results, the ones with the numbers and the word or the numbers. That start the string

right. And I'm going to search those results for any patterns that have Http

and anything after it. Right? So the pattern needs to contain, or the string needs to contain, this pattern.

followed by any character that occurs any number of time.

Right? So that's to extract the entire URL.

Alright. So we see here in this 1st entry the 115th character.

So if we were to count a hundred 15, that would be this H in this URL here.

right? So we've extracted the links, and then we can again use the Reg matches function to extract those corresponding links from the strings

right? So here you could see we have each of these.

And now we can store those links in, say, a data frame, and then search those to verify the accuracy of numbers, and so on.