Regular Expressions Describe Patterns in Character Strings

Regular expressions:
- a concise mechanism (language?) for describing patterns in characters strings

To work with regular expressions, we'll use **`str_subset()`** from the **stringr** package

```
str_subset(vector_of_character_strings, regular_expression)
```

returns a vector containing the character strings in vector_of_character_strings that match the pattern described by regular_expression

```
install.packages("stringr")
library(stringr)
```

Workshop Data

US Birth Names, by sex and year (1882-2012), if count >= 500

- data read in as a dataframe containing a single vector of character strings

```
yob <- read.csv(file="yob_notes.csv", head=TRUE, stringsAsFactors=FALSE)
notes <- as.character(yob$notes)
```

```
head(notes,12)
```

```
 [1] "name: Mary, sex: F, year: 1882, count: 8148, rank: 1, format: Xxxx, top-10 at least once / earliest 500+ count: 1882"
 [2] "name: Anna, sex: F, year: 1882, count: 3143, rank: 2, format: Xxxx, top-10 at least once / earliest 500+ count: 1882"
 [3] "name: Emma, sex: F, year: 1882, count: 2303, rank: 3, format: Xxxx, top-10 at least once / earliest 500+ count: 1882"
 [4] "name: Elizabeth, sex: F, year: 1882, count: 2187, rank: 4, format: Xxxxxxxxx, top-10 at least once / earliest 500+ count: 1882"
 [5] "name: Minnie, sex: F, year: 1882, count: 2004, rank: 5, format: Xxxxxx, top-10 at least once / earliest 500+ count: 1882"
 [6] "name: Margaret, sex: F, year: 1882, count: 1821, rank: 6, format: Xxxxxxxx, top-10 at least once / earliest 500+ count: 1882"
 [7] "name: Ida, sex: F, year: 1882, count: 1673, rank: 7, format: Xxx, top-10 at least once / earliest 500+ count: 1882"
 [8] "name: Alice, sex: F, year: 1882, count: 1542, rank: 8, format: Xxxxx, top-10 at least once / earliest 500+ count: 1882"
 [9] "name: Bertha, sex: F, year: 1882, count: 1508, rank: 9, format: Xxxxxx, top-10 at least once / earliest 500+ count: 1882"
[10] "name: Annie, sex: F, year: 1882, count: 1492, rank: 10, format: Xxxxx, top-10 at least once / earliest 500+ count: 1882"
[11] "name: Clara, sex: F, year: 1882, count: 1490, rank: 11, format: Xxxxx, top-10 at least once / earliest 500+ count: 1882"
[12] "name: Sarah, sex: F, year: 1882, count: 1410, rank: 12, format: Xxxxx, top-10 at least once / earliest 500+ count: 1882"
```

Simplest Patterns

Simplest regular expressions match exact character strings:

```
str_subset(notes, "rank: 1,")
```

```
[1] "name: Mary, sex: F, year: 1882, count: 8148, rank: 1, format: Xxxx, top-10 at least once / earliest 500+ count: 1882"
[2] "name: Mary, sex: F, year: 1883, count: 8012, rank: 1, format: Xxxx, top-10 at least once / earliest 500+ count: 1882"
[3] "name: Mary, sex: F, year: 1884, count: 9217, rank: 1, format: Xxxx, top-10 at least once / earliest 500+ count: 1882"
                                                                                                                        .
                                                                                                                        .
                                                                                                                        .
[129] "name: Isabella, sex: F, year: 2010, count: 22856, rank: 1, format: Xxxxxxxx, top-10 at least once / earliest 500+ count: 1992"
[130] "name: Sophia, sex: F, year: 2011, count: 21780, rank: 1, format: Xxxxxx, top-10 at least once / earliest 500+ count: 1914"
[131] "name: Sophia, sex: F, year: 2012, count: 22158, rank: 1, format: Xxxxxx, top-10 at least once / earliest 500+ count: 1914"
[132] "name: John, sex: M, year: 1882, count: 9557, rank: 1, format: Xxxx, top-10 at least once / earliest 500+ count: 1882"
[133] "name: John, sex: M, year: 1883, count: 8894, rank: 1, format: Xxxx, top-10 at least once / earliest 500+ count: 1882"
[134] "name: John, sex: M, year: 1884, count: 9387, rank: 1, format: Xxxx, top-10 at least once / earliest 500+ count: 1882"
                                                                                                                        .
                                                                                                                        .
                                                                                                                        .
[260] "name: Jacob, sex: M, year: 2010, count: 22052, rank: 1, format: Xxxxx, top-10 at least once / earliest 500+ count: 1912"
[261] "name: Jacob, sex: M, year: 2011, count: 20269, rank: 1, format: Xxxxx, top-10 at least once / earliest 500+ count: 1912"
[262] "name: Jacob, sex: M, year: 2012, count: 18899, rank: 1, format: Xxxxx, top-10 at least once / earliest 500+ count: 1912"
```

Simplest Patterns

Simplest regular expressions match exact character strings

What happens if regular expression does not include the comma?

```
str_subset(notes, "rank: 1")
```

Why??

Match Any Single Character

Within a regular expression, use meta character dot (**.**) to match any single character, except the newline character

```
str_subset(notes, "name: .ay,")

str_subset(notes, "name: A..,")
```

How to match the character dot (.) ?

- use backslashes (****) as an “escape” sequence to escape the special meaning of dot

```
s1 <- c("Hello.",  "How are you?", "Happy to be learning!")

str_subset(s1, ".")
str_subset(s1, "\\.")
```

Anchors

To indicate that a pattern must appear at the **start** of a character string, use **^**

To indicate that a pattern must appear at the **end** of a character string, use **$**

Which of these regular expressions find all names that first have a 500+ count in 2012?

```
str_subset(notes, "2012")

str_subset(notes, "2012$")
```

Match Any One of a Set of Characters

A character class, indicated by a pair of square brackets, **[** and **]**, is used to indicate that a pattern must contain any one of the characters contained within the square brackets.

```
str_subset(notes, "rank: [123],")
```

A dash (**-**) between two characters inside a character class indicates a range of characters:

```
str_subset(notes, "201[0-2]$")
```

To match any character except the characters specified between square brackets, use **^** as the first character inside the character class.

```
str_subset(notes, "name: [^A-LOU]..,")
```

Exercises

Find the one name contained in the notes vector that has a **q** as the 5th letter, and that **q** is <u>not</u> followed by a **u**.

Find the two names stored in the notes vector that have a **q** as the 5th letter, and a **u** as the 6th letter.

Exercise: Solution

Find the one name contained in the notes vector that has a **q** as the 5th letter, and that **q** is not followed by a **u**.

```
str_subset(notes, "name: ....q[^u]")
```

Find the two names stored in the notes vector that have a **q** as the 5th letter, and a **u** as the 6th letter.

```
str_subset(notes, "name: ....qu")
```

Match Any One of a Set of Characters

```
str_subset(notes, "name: Sophia,")

str_subset(notes, "name: Sophia,") %>% str_subset("rank: [1-9],")

str_subset(notes, "name: Sophia,") %>% str_subset("rank: [:digit:],")

str_subset(notes, "name: Sophia,") %>% str_subset("rank: [:digit:][:digit:],")

str_subset(notes, "name: Sophia,") %>% str_subset("rank: [:digit:][:digit:][:digit:],")
```

Character Class Shorthands

Shorthand	Meaning
`[:digit:]`	single digit
`[:alpha:]`	single letter
`[:lower:]`	single lowercase letter
`[:upper:]`	single uppercase letter
`[:alnum:]`	single letter or digit
`[:punct:]`	single punctuation mark
`[:graph:]`	single letter, digit or punctuation mark

Alternation

Vertical bar, **|**, can be used to match one or more alternative patterns.

```
str_subset(notes, "year: 201.,") %>% str_subset("name: So(ph|f)ia,")

str_subset(notes, "year: 1956,") %>% str_subset("name: Al(a|la|le)n,")
```

Vertical bar has very low precedence. If parentheses are not used, alternatives are different:

```
str_subset(notes, "year: 201.,") %>% str_subset("name: Soph|fia")
```

- character strings that contain “Soph” or “fia,”
- matches name Sophie in addition to Sophia and Sofia

```
str_subset(notes, "year: 1956,") %>% str_subset("name: Ala|la|len")
```

- character strings that contain “Ala” or “la” or “len,”
- matches names Pamela, Ellen, and many others in addition to Alan, Allan, Allen

Character Class Shorthands and Alternation Exercises

List all lines for the name Mary where the name has a single-digit rank (1-9).

List all lines for the name Mary where the name has rank between 90 and 99.

List all lines that show names in 2012 where the count is at least 10,000.

Using alternation, list all lines for the name Stacie, Stacey or Stacy during the 1980's.

Using alternation, list all lines for Mary or Marie where the name has a single-digit rank (1-9).

Exercises: Solutions

List all lines for the name Mary where the name has a single-digit rank (1-9).

```
str_subset(notes, "name: Mary,")  %>%  str_subset("rank: [1-9],")
```

List all lines for the name Mary where the name has rank between 90 and 99.

```
str_subset(notes, "name: Mary,")  %>%  str_subset("rank: 9[:digit:],")
```

List all lines that show names in 2012 where the count is at least 10,000.

```
str_subset(notes, "year: 2012") %>% str_subset("count: [1-9][:digit:][:digit:][:digit:][:digit:]")
```

Using alternation, list all lines for the name Stacie, Stacey or Stacy during the 1980's.

```
str_subset(notes, "Stac(ie|ey|y), sex: F, year: 198[0-9]")
```

Using alternation, list all lines for Mary or Marie where the name has a single-digit rank (1-9).

```
str_subset(notes, "name: Mar(y|ie),")  %>%  str_subset("rank: [:digit:],")
```

Repetition

+ one or more of the preceding character

```
str_subset(notes, "year: 201.,") %>% str_subset("Al+ison,")
```

? zero or one of the preceding character

```
str_subset(notes, "[A-Z]ou?,")
```

***** zero or more of the preceding character

```
str_subset(notes, "year: 2012") %>% str_subset("[A-Z].*ee")
```

```
str_subset(notes, "[A-Z].*ee.*year: 2012")
```

Repetition Exercises

Use a repetition meta character (**+** or **?** or *****) to extract the rows that contain the names "Ana" and "Anna" in the 1950s.

Use a repetition meta character (**+** or **?** or *****) to determine whether there are more male names or more female names that start with "N" and end with **"**y**"**

Use a repetition meta character (**+** or **?** or *****) to see whether the names "Jo**"** and "Joe" appear in the notes vector.

Repetition Exercises: Solutions

Use a repetition meta character (**+** or **?** or *****) to extract the rows that contain the names "Ana" and "Anna" in the 1950s.

```
str_subset(notes, "year: 195[0-9]") %>% str_subset("An+a,")
```

Use a repetition meta character (**+** or **?** or *****) to determine whether there are more male names or more female names that start with "N" and end with "y“

```
str_subset(notes, "name: N.*y,")
```

Use a repetition meta character (**+** or **?** or *****) to see whether the names "Jo**"** and "Joe" appear in the notes vector.

```
str_subset(notes, "name: Joe?,")
```

Repetition

{n} exactly **n** of the preceding character

```
str_subset(notes, "[A-Z][aeiou]{3}[^aeiou].*year: 2012")
```

why needed?

{n,} **n** or more of the preceding character

```
str_subset(notes, "[A-Z][aeiou]{2,}.*year: 2012")
```

{m,n} between **m** and **n** of the preceding character

```
str_subset(notes, "year: 2012.*format: Xx{7,8}[^x]")
```

why needed?

Repetition Exercises

List all rows containing names in 2012 that start with a vowel followed by exactly 2 consonants. Examples: "Edgar" and "Amy"

List all rows containing names in 2012 that start with a vowel followed by 3 or more consonants in a row and then a vowel. Examples: "Alyssa" and "Ashton"

Repetition Exercises: Solutions

List all rows containing names in 2012 that start with a vowel followed by exactly 2 consonants. Examples: "Edgar" and "Amy"

```
str_subset(notes, "year: 2012") %>% str_subset("[AEIOU][^aeiou,]{2}[aeiou,]")
```

List all rows containing names in 2012 that start with a vowel followed by 3 or more consonants in a row and then a vowel. Examples: "Alyssa" and "Ashton"

```
str_subset(notes, "year: 2012") %>% str_subset("[AEIOU][^aeiou,]{3,}[aeiou]")
```

Grouping and Back Referencing

Parentheses can be used within regular expressions to refer back to part of a pattern later on.

```
str_subset(notes, "year: 1969") %>% str_subset("name: .(.)(.)\\2\\1")
```

[1] "name: **Cassandra**, sex: F, year: 1969, count: 2164, rank: 151, format: Xxxxxxxxx, earliest 500+ count: 1949"
[2] "name: **Kelley**, sex: F, year: 1969, count: 1685, rank: 191, format: Xxxxxx, never in top-100 / earliest 500+ count: 1958"
[3] "name: **Billie**, sex: F, year: 1969, count: 823, rank: 312, format: Xxxxxx, earliest 500+ count: 1916"
[4] "name: **Lillian**, sex: F, year: 1969, count: 773, rank: 324, format: Xxxxxxx, top-10 at least once / earliest 500+ count: 1882“
[5] "name: **William**, sex: M, year: 1969, count: 37647, rank: 6, format: Xxxxxxx, top-10 at least once / earliest 500+ count: 1882“
[6] "name: **Kenneth**, sex: M, year: 1969, count: 17557, rank: 23, format: Xxxxxxx, earliest 500+ count: 1906"
[7] "name: **Jeffery**, sex: M, year: 1969, count: 6745, rank: 48, format: Xxxxxxx, earliest 500+ count: 1946"
[8] "name: **Willie**, sex: M, year: 1969, count: 3048, rank: 102, format: Xxxxxx, earliest 500+ count: 1882"
[9] "name: **Jesse**, sex: M, year: 1969, count: 2381, rank: 128, format: Xxxxx, earliest 500+ count: 1882"
[10] "name: **Terrence**, sex: M, year: 1969, count: 1219, rank: 205, format: Xxxxxxxx, never in top-100 / earliest 500+ count: 1942"
[11] "name: **Jimmie**, sex: M, year: 1969, count: 828, rank: 254, format: Xxxxxx, earliest 500+ count: 1914"

Grouping and Back Referencing

Parentheses can be used within regular expressions to refer back to part of a pattern later on.

```
str_subset(notes, "year: 1969") %>% str_subset("name: .(.).\\1.\\1")
```

[1] "name: **T**a**m**a**r**a, sex: F, year: 1969, count: 4652, rank: 78, format: Xxxxxx, earliest 500+ count: 1952"

```
str_subset(notes, "year: 2012") %>% str_subset("name: .([aeiou]n).*\\1, sex")
```

[1] "name: **L**o**n**d**on**, sex: F, year: 2012, count: 3179, rank: 94, format: Xxxxxx, earliest 500+ count: 2005“

[2] "name: **J**e**n**s**en**, sex: M, year: 2012, count: 516, rank: 509, format: Xxxxxx, never in top-100 / earliest 500+ count: 2012"

Grouping and Back Referencing Exercises

List all rows containing names in 2012 that have a vowel as the second letter and end with that same vowel. For example: Romeo. There are 62 girls names that match this pattern, and 12 boys names.

List all rows containing names in years 2000-2012 that have a vowel as the second letter, and end with that same vowel and whatever letter immediately follows that vowel. For example: Jensen.
This can be done using one group or two. Try to match these 21 rows both ways.

Grouping and Back Referencing Exercises: Solutions

List all rows containing names in 2012 that have a vowel as the second letter and end with that same vowel. For example: Romeo. There are 62 girls names that match this pattern, and 12 boys names.

```
str_subset(notes, "year: 2012") %>% str_subset("name: .([aeiou]).*\\1, sex")
```

List all rows containing names in years 2000-2012 that have a vowel as the second letter, and end with that same vowel and whatever letter immediately follows that vowel. For example: Jensen.
This can be done using one group or two. Try to match these 21 rows both ways.

```
str_subset(notes, "name: .([aeiou])(.).*\\1\\2, sex.*year: 20")
```

```
str_subset(notes, "name: .([aeiou].).*\\1, sex.*year: 20")
```

Explicitly Calling **`regex`** Function

Character string patterns are automatically wrapped into a call to **`regex`** function.

```
str_subset(notes, "[A-Z]ou?,")
```

is shorthand for:

```
str_subset(notes, regex("[A-Z]ou?,"))
```

`regex` function provides an argument to ignore case:

`ignore_case = TRUE` allows characters to match either their uppercase or lowercase forms

```
str_subset(notes, regex("[jlr]ay...,.*year: 2012", ignore_case=TRUE))
```

produces same subset of character strings as

```
str_subset(notes, "[JjLlRr]ay...,.*year: 2012")
```

More Exercises

List all lines for 1900 where name had been in "top-10" at least once, or name had never been in "top-100"

List lines for boys names in 1900 where name had been in "top-10" at least once, or name had never been in "top-100"

List lines that show all names that first appeared in 1910

List lines for 1999 where names start with J, 3rd and 4th letters of the name are the same, and the 2nd letter appears somewhere after that.

List all lines for boys names that have a single digit rank and first appeared at the start of a decade (1890, 1900, 1910, ... 2010)

List all lines for girls names that start with D, have a double digit rank and first appeared in the 1960s.

List all lines that contain names with 11 letters. Try to do this using a repetition indicator.

List all lines for 1925 that contain names that are 2 or 3 letters.

List all lines where the count value contains 5 digts and is a palindrome, for example: 10901, 21612, 18981.

More Exercises: Solutions

List all lines for 1900 where name had been in "top-10" at least once, or name had never been in "top-100"
```
str_subset(notes, "year: 1900,.*-")
```

List lines for boys names in 1900 where name had been in "top-10" at least once, or name had never been in "top-100"
```
str_subset(notes, "M, year: 1900,.*-")
```

List lines that show all names that first appeared in 1910
```
str_subset(notes, "2010$")
```

List lines for 1999 where names start with J, 3rd and 4th letters of the name are the same, and the 2nd letter appears somewhere after that.
```
str_subset(notes, "name: J(.)(.)\\2.*\\1.*, sex.*year: 1999")
```

List all lines for boys names that have a single digit rank and first appeared at the start of a decade (1890, 1900, 1910, ... 2010)
```
str_subset(notes, "M,.*rank: [:digit:],.* ...0$")
```

List all lines for girls names that start with D, have a double digit rank and first appeared in the 1960s.
```
str_subset(notes, "D.*F,.*rank: [:digit:][:digit:],.*196.$")
```

List all lines that contain names with 11 letters. Try to do this using a repetition indicator.
```
str_subset(notes, "Xxxxxxxxxxx,")
str_subset(notes, "Xx{10},")
```

List all lines for 1925 that contain names that are 2 or 3 letters.
```
str_subset(notes, "year: 1925.*Xxx?,")
```

List all lines where the count value contains 5 digits and is a palindrome, for example: 10901, 21612, 18981.
```
str_subset(notes, "count: ([:digit:])([:digit:])([:digit:])\\2\\1")
```

Additional Resources

R Documentation:

Regular Expressions
Online at https://stringr.tidyverse.org/articles/regular-expressions.html

Regular Expressions as used in R
Online at https://stat.ethz.ch/R-manual/R-devel/library/base/html/regex.html

Book:

R for Data Science (Chapter 4), by Hadley Wickham and Garrett Grolemund, 2016.
Online at http://r4ds.had.co.nz/strings.html

Course Notes:

STAT 545 Regular Expressions in R, by Gloria Li and Jenny Gryan, October 19, 2014.
Online at http://stat545.com/block022_regular-expression.html

Cheat Sheet:

String manipulation with stringr:: Cheet Sheet by Rstudio, 2017-10.
Online at https://www.rstudio.com/resources/cheatsheets/#stringr