Regular Expressions With stringr

Outline

Regular Expressions (regex's)

Outline

Regular Expressions (regex's)

Meta, escape, and special characters

Outline

Regular Expressions (regex's)

Meta, escape, and special characters

Character Classes

Outline

Regular Expressions (regex's)

Meta, escape, and special characters

Character Classes

POSIX classes

Outline

Regular Expressions (regex's)

Meta, escape, and special characters

Character Classes

POSIX classes

Groups and alternatives

Outline

Regular Expressions (regex's)

Meta, escape, and special characters

Character Classes

POSIX classes

Groups and alternatives

Back referencing

Outline

Regular Expressions (regex's)

Meta, escape, and special characters

Character Classes

POSIX classes

Groups and alternatives

Back referencing

Data cleaning example

Motivation

Suppose I am trying to scrape people’s email from their website to send them spam!

Motivation

Suppose I am trying to scrape people's email from their website to send them spam!

People generally don't appreciate that and so try to make it hard for the computer to "scrape" emails!

Motivation

Suppose I am trying to scrape people's email from their website to send them spam!

People generally don't appreciate that and so try to make it hard for the computer to "scrape" emails!

```
emails <- c(
"my email is foo@bar.com.",
"email me at foo AT bar.io!",
"foo @ bar DOT edu is where i'm at",
"you can't scrape foo AT bar DOT info",
"shh, foo AT bar . gov is my email address"
)
```

Motivation

Suppose I am trying to scrape people's email from their website to send them spam!

People generally don't appreciate that and so try to make it hard for the computer to "scrape" emails!

```
emails <- c(
"my email is foo@bar.com.",
"email me at foo AT bar.io!",
"foo @ bar DOT edu is where i'm at",
"you can't scrape foo AT bar DOT info",
"shh, foo AT bar . gov is my email address"
)
```

How can I write one piece of code to scrape all of this? Is it possible?

Regex

A *regular expression* (regex for short) is a special text string for describing a search pattern

Regex

A *regular expression* (regex for short) is a special text string for describing a search pattern

All stringr functions work with regex patterns. Before we were only directly matching, but we can do way more than that!

Regex

A *regular expression* (regex for short) is a special text string for describing a search pattern

All stringr functions work with regex patterns. Before we were only directly matching, but we can do way more than that!

```
"the green grass flows" %>% str_extract("g")
```

Regex

A *regular expression* (regex for short) is a special text string for describing a search pattern

All stringr functions work with regex patterns. Before we were only directly matching, but we can do way more than that!

```
"the green grass flows" %>% str_extract("g")
```

This is a regex!

Regex

A *regular expression* (regex for short) is a special text string for describing a search pattern

All stringr functions work with regex patterns. Before we were only directly matching, but we can do way more than that!

```
"the green grass flows" %>% str_extract("g")
```

This is a regex!

Regex's are interpreted by a regex engine. They are basically their own programming language!

Metacharacters

In regex’s there are certain characters that have special meanings. They are called metacharacters!

Metacharacters

In regex's there are certain characters that have special meanings. They are called metacharacters!

. * + - ? ! ^ | [] () {} \ = $:

Metacharacters

In regex's there are certain characters that have special meanings. They are called metacharacters!

. * + - ? ! ^ | [] () {} \ = $:

For instance, the . is a wild character that matches almost anything!

Metacharacters

In regex's there are certain characters that have special meanings. They are called metacharacters!

. * + - ? ! ^ | [] () {} \ = $:

For instance, the . is a wild character that matches almost anything!

```
"hit hat hot him hilt" %>% str_extract_all("h.t")
```

Metacharacters

In regex's there are certain characters that have special meanings. They are called metacharacters!

. * + - ? ! ^ | [] () {} \ = $:

For instance, the . is a wild character that matches almost anything!

```
"hit hat hot him hilt" %>% str_extract_all("h.t")
```

```
[[1]]
[1] "hit" "hat" "hot"
```

Escaped Characters

Since metacharacters have special meanings, there has to be a way to match a literal version

Escaped Characters

Since metacharacters have special meanings, there has to be a way to match a literal version

Ex. How would I match a . since it is a metacharacter?

Escaped Characters

Since metacharacters have special meanings, there has to be a way to match a literal version

Ex. How would I match a . since it is a metacharacter?

A \ escapes a character which makes it literal. In R, you must use two backslashes (\\) to escape characters

Escaped Characters

Since metacharacters have special meanings, there has to be a way to match a literal version

Ex. How would I match a . since it is a metacharacter?

A \ escapes a character which makes it literal. In R, you must use two backslashes (\\) to escape characters

```
"225 2.5" %>% str_extract("2.5")
```

Escaped Characters

Since metacharacters have special meanings, there has to be a way to match a literal version

Ex. How would I match a . since it is a metacharacter?

A \ escapes a character which makes it literal. In R, you must use two backslashes (\\) to escape characters

```
"225 2.5" %>% str_extract("2.5")
```

```
[1] "225"
```

Escaped Characters

Since metacharacters have special meanings, there has to be a way to match a literal version

Ex. How would I match a . since it is a metacharacter?

A \ escapes a character which makes it literal. In R, you must use two backslashes (\\) to escape characters

```
"225 2.5" %>% str_extract("2.5")
```

```
[1] "225"
```

```
"225 2.5" %>% str_extract("2\\.5")
```

Escaped Characters

Since metacharacters have special meanings, there has to be a way to match a literal version

Ex. How would I match a . since it is a metacharacter?

A \ escapes a character which makes it literal. In R, you must use two backslashes (\\) to escape characters

```
"225 2.5" %>% str_extract("2.5")
```

```
[1] "225"
```

```
"225 2.5" %>% str_extract("2\\.5")
```

```
[1] "2.5"
```

Special characters

When you escape non-metacharacters with one \ , they get a special meaning as single characters.

Special characters

When you escape non-metacharacters with one \ , they get a special meaning as single characters.

For instance.

- \n is a new line
- \t is a tab
- \e is escape
- etc.

Special characters

When you escape non-metacharacters with one \ , they get a special meaning as single characters.

For instance.

- \n is a new line
- \t is a tab
- \e is escape
- etc.

```
cat("a\nb")
```

Special characters

When you escape non-metacharacters with one \ , they get a special meaning as single characters.

For instance.

- \n is a new line
- \t is a tab
- \e is escape
- etc.

```
cat("a\nb")
```

```
a
b
```

Character classes

A *character class* is a list of characters enclosed between [and] which matches any single character in that list

Character classes

A *character class* is a list of characters enclosed between [and] which matches any single character in that list

```
"I use summarize instead of summarise" %>%
str_extract_all("summari[sz]e")
```

Character classes

A *character class* is a list of characters enclosed between [and] which matches any single character in that list

```
"I use summarize instead of summarise" %>%
str_extract_all("summari[sz]e")
```

```
[[1]]
[1] "summarize" "summarise"
```

Character classes

A *character class* is a list of characters enclosed between [and] which matches any single character in that list

```
"I use summarize instead of summarise" %>%
str_extract_all("summari[sz]e")
```

```
[[1]]
[1] "summarize" "summarise"
```

You can specify ranges in a character class using -

Character classes

A *character class* is a list of characters enclosed between [and] which matches any single character in that list

```
"I use summarize instead of summarise" %>%
str_extract_all("summari[sz]e")
```

```
[[1]]
[1] "summarize" "summarise"
```

You can specify ranges in a character class using -

```
"Only 5 students failed last semester!" %>%
    str_extract("[0-9]")
```

Character classes

A *character class* is a list of characters enclosed between [and] which matches any single character in that list

```
"I use summarize instead of summarise" %>%
str_extract_all("summari[sz]e")
```

```
[[1]]
[1] "summarize" "summarise"
```

You can specify ranges in a character class using -

```
"Only 5 students failed last semester!" %>%
    str_extract("[0-9]")
```

```
[1] "5"
```

Character classes

Metacharacters often become literal inside character classes

Character classes

Metacharacters often become literal inside character classes

```
"225 2.5" %>% str_extract("2[.]5")
```

Character classes

Metacharacters often become literal inside character classes

```
"225 2.5" %>% str_extract("2[.]5")
```

```
[1] "2.5"
```

Character classes

Metacharacters often become literal inside character classes

```
"225 2.5" %>% str_extract("2[.]5")
```

```
[1] "2.5"
```

There are only a couple of exceptions to the rule (^ - \ )

Character classes

Metacharacters often become literal inside character classes

```
"225 2.5" %>% str_extract("2[.]5")
```

```
[1] "2.5"
```

There are only a couple of exceptions to the rule (^ - \ )

^ at the beginning of a character class says match anything except that (negates)

Character classes

Metacharacters often become literal inside character classes

```
"225 2.5" %>% str_extract("2[.]5")
```

```
[1] "2.5"
```

There are only a couple of exceptions to the rule (^ - \ )

^ at the beginning of a character class says match anything except that (negates)

```
"hat hit hot hut" %>% str_extract_all("h[^a]t")
```

Character classes

Metacharacters often become literal inside character classes

```
"225 2.5" %>% str_extract("2[.]5")
```

```
[1] "2.5"
```

There are only a couple of exceptions to the rule (^ - \ )

^ at the beginning of a character class says match anything except that (negates)

```
"hat hit hot hut" %>% str_extract_all("h[^a]t")
```

```
[[1]]
[1] "hit" "hot" "hut"
```

Character Classes Shortcuts

Some escaped literals correspond to character classes

	English	Equivalent to
\d \D	Single Digit Not Single Digit	[0-9] [^0-9]
\w \W	Word Character Not Word Character	[0-9a-zA-Z] [^0-9a-zA-Z]
\s \S	Whitespace Not Whitespace	[\n\t\r] [^\n\t\r]

Quick Character Classes

```
string <- c("My office phone number is 717-477-1468",
            "The department chair's number is 717-477-1450")
```

Quick Character Classes

```
string <- c("My office phone number is 717-477-1468",
            "The department chair's number is 717-477-1450")


string %>%
    str_extract_all("\\d\\d\\d-\\d\\d\\d-\\d\\d\\d\\d")
```

Quick Character Classes

```
string <- c("My office phone number is 717-477-1468",
            "The department chair's number is 717-477-1450")
```

```
string %>%
    str_extract_all("\\d\\d\\d-\\d\\d\\d-\\d\\d\\d\\d")
```

```
[[1]]
[1] "717-477-1468"

[[2]]
[1] "717-477-1450"
```

POSIX classes

Credit: David Kahle

	English	Equivalent to
[:alnum:]	Alphanumeric	[:alpha:] and [:digit:]
[:alpha:]	Alphabetical	[:lower:] and [:upper:]
[:digit:]	Single digit	[0-9]
[:graph:]	Graphical	[:alnum:] and [:punct:]
[:lower:]	Lower case	[a-z]
[:punct:]	Punctuation	! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { | } ~.
[:space:]	Space	Tab, newline, etc.
[:upper:]	Upper case	[A-Z]

Credit: David Kahle

regex quantifiers

```
string <- "bb bab baab baaab"
```

Regex	Match _____ a's	Example
a?	Zero or one a is optional	`string %>% str_view_all("ba?")` `"bb bab baab baaab"`

Credit: David Kahle

regex quantifiers

```
string <- "bb bab baab baaab"
```

Regex	Match _____ a's	Example
a?	Zero or one a is optional	`string %>% str_view_all("ba?")` `"bb bab baab baaab"`
a*	Zero or more a is optional	`string %>% str_view_all("ba*")` `"bb bab baab baaab"`

Credit: David Kahle

regex quantifiers

```
string <- "bb bab baab baaab"
```

Regex	Match _____ a's	Example
a?	Zero or one a is optional	`string %>% str_view_all("ba?")` `"bb bab baab baaab"`
a*	Zero or more a is optional	`string %>% str_view_all("ba*")` `"bb bab baab baaab"`
a+	One or more	`string %>% str_view_all("ba+")` `"bb bab baab baaab"`

Credit: David Kahle

regex quantifiers

```
string <- "bb bab baab baaab"
```

Regex	Match _____ a's	Example
a?	Zero or one a is optional	`string %>% str_view_all("ba?")` `"bb bab baab baaab"`
a*	Zero or more a is optional	`string %>% str_view_all("ba*")` `"bb bab baab baaab"`
a+	One or more	`string %>% str_view_all("ba+")` `"bb bab baab baaab"`
a{n}	Exactly n	`string %>% str_view_all("ba{2}")` `"bb bab baab baaab"`

Credit: David Kahle

regex quantifiers

```
string <- "bb bab baab baaab"
```

Regex	Match _____ a's	Example
a?	Zero or one a is optional	`string %>% str_view_all("ba?")` `"bb bab baab baaab"`
a*	Zero or more a is optional	`string %>% str_view_all("ba*")` `"bb bab baab baaab"`
a+	One or more	`string %>% str_view_all("ba+")` `"bb bab baab baaab"`
a{n}	Exactly n	`string %>% str_view_all("ba{2}")` `"bb bab baab baaab"`
a{n,}	At least n	`string %>% str_view_all("ba{2,}")` `"bb bab baab baaab"`

Credit: David Kahle

regex quantifiers

```
string <- "bb bab baab baaab"
```

Regex	Match _____ a's	Example
a?	Zero or one a is optional	`string %>% str_view_all("ba?")` `"bb bab baab baaab"`
a*	Zero or more a is optional	`string %>% str_view_all("ba*")` `"bb bab baab baaab"`
a+	One or more	`string %>% str_view_all("ba+")` `"bb bab baab baaab"`
a{n}	Exactly n	`string %>% str_view_all("ba{2}")` `"bb bab baab baaab"`
a{n,}	At least n	`string %>% str_view_all("ba{2,}")` `"bb bab baab baaab"`
a{n,m}	Between n and m	`string %>% str_view_all("ba{1,2}")` `"bb bab baab baaab"`

Quantifiers

```
string <- c("My office phone number is 717-477-1468",
            "The department chair's number is 717-477-1450")
```

Quantifiers

```
string <- c("My office phone number is 717-477-1468",
            "The department chair's number is 717-477-1450")

string %>%
    str_extract_all("\\d{3}-\\d{3}-\\d{4}")
```

Quantifiers

```
string <- c("My office phone number is 717-477-1468",
            "The department chair's number is 717-477-1450")
```

```
string %>%
    str_extract_all("\\d{3}-\\d{3}-\\d{4}")
```

```
[[1]]
[1] "717-477-1468"

[[2]]
[1] "717-477-1450"
```

Quantifiers

```
string <- c("My office phone number is 717-477-1468",
            "The department chair's number is 717-477-1450")
```

```
string %>%
    str_extract_all("\\d{3}-\\d{3}-\\d{4}")
```

```
[[1]]
[1] "717-477-1468"

[[2]]
[1] "717-477-1450"
```

```
string %>%
    str_extract_all("[0-9]+-[0-9]+-[0-9]+")
```

Quantifiers

```
string <- c("My office phone number is 717-477-1468",
            "The department chair's number is 717-477-1450")
```

```
string %>%
    str_extract_all("\\d{3}-\\d{3}-\\d{4}")
```

```
[[1]]
[1] "717-477-1468"

[[2]]
[1] "717-477-1450"
```

```
string %>%
    str_extract_all("[0-9]+-[0-9]+-[0-9]+")
```

```
[[1]]
[1] "717-477-1468"

[[2]]
[1] "717-477-1450"
```

Groups And Alternatives

Groups are made with the metacharacters (and). They group characters together to act as one single unit

Groups And Alternatives

Groups are made with the metacharacters (and). They group characters together to act as one single unit

```
"ld lad led lead" %>% str_extract_all("l(ea)?d")
```

Groups And Alternatives

Groups are made with the metacharacters (and). They group characters together to act as one single unit

```
"ld lad led lead" %>% str_extract_all("l(ea)?d")
```

```
[[1]]
[1] "ld"   "lead"
```

Groups And Alternatives

Groups are made with the metacharacters (and). They group characters together to act as one single unit

```
"ld lad led lead" %>% str_extract_all("l(ea)?d")
```

```
[[1]]
[1] "ld"   "lead"
```

```
"ld lad led lead" %>% str_extract_all("lea?d")
```

Groups And Alternatives

Groups are made with the metacharacters (and). They group characters together to act as one single unit

```
"ld lad led lead" %>% str_extract_all("l(ea)?d")
```

```
[[1]]
[1] "ld"   "lead"
```

```
"ld lad led lead" %>% str_extract_all("lea?d")
```

```
[[1]]
[1] "led"  "lead"
```

Groups And Alternatives

The metacharacter | corresponds to or and is very powerful because it is often the case that you want to search for different patterns all at once

Groups And Alternatives

The metacharacter | corresponds to or and is very powerful because it is often the case that you want to search for different patterns all at once

With this (and all other stuff we learned) we can now scrape some emails!

Groups And Alternatives

The metacharacter | corresponds to or and is very powerful because it is often the case that you want to search for different patterns all at once

With this (and all other stuff we learned) we can now scrape some emails!

```
emails <- c(
"my email is foo@bar.com.",
"email me at foo AT bar.io!",
"foo @ bar DOT edu is where i'm at",
"you can't scrape foo AT bar DOT info",
"shh, foo AT bar . gov is my email address"
)
```

How do we do it?

Groups And Alternatives

```
emails <- c(
"my email is foo@bar.com.",
"email me at foo AT bar.io!",
"foo @ bar DOT edu is where i'm at",
"you can't scrape foo AT bar DOT info",
"shh, foo AT bar . gov is my email address"
)
```

How can I detect the @ or at?

How can I detect the . or dot?

How can I detect the domain name?

How do I put it all together?

Groups And Alternatives

```
emails <- c(
"my email is foo@bar.com.",
"email me at foo AT bar.io!",
"foo @ bar DOT edu is where i'm at",
"you can't scrape foo AT bar DOT info",
"shh, foo AT bar . gov is my email address"
)
```

Groups And Alternatives

```
emails <- c(
"my email is foo@bar.com.",
"email me at foo AT bar.io!",
"foo @ bar DOT edu is where i'm at",
"you can't scrape foo AT bar DOT info",
"shh, foo AT bar . gov is my email address"
)

emails %>% str_to_lower() %>%
    str_extract_all("\\w+( *@ *| *at *)\\w+( *\\. *| *dot *)
(com|io|info|edu|gov)")
```

Groups And Alternatives

```
emails <- c(
"my email is foo@bar.com.",
"email me at foo AT bar.io!",
"foo @ bar DOT edu is where i'm at",
"you can't scrape foo AT bar DOT info",
"shh, foo AT bar . gov is my email address"
)

emails %>% str_to_lower() %>%
    str_extract_all("\\w+( *@ *| *at *)\\w+( *\\. *| *dot *)
(com|io|info|edu|gov)")
```

glue is nice for splitting this long string into smaller, more manageable parts

Back Referencing

Groups can also be used for back-referencing (matching the same pattern previously found)

Back Referencing

Groups can also be used for back-referencing (matching the same pattern previously found)

```
"I kicked the the ball" %>% str_extract_all("(\\w+) \\1")
```

Back Referencing

Groups can also be used for back-referencing (matching the same pattern previously found)

```
"I kicked the the ball" %>% str_extract_all("(\\w+) \\1")
```

```
[[1]]
[1] "the the"
```

Back Referencing

Groups can also be used for back-referencing (matching the same pattern previously found)

```
"I kicked the the ball" %>% str_extract_all("(\\w+) \\1")
```

```
[[1]]
[1] "the the"
```

```
"I kicked the the ball" %>% str_replace_all("(\\w+) \\1", "\\1")
```

Back Referencing

Groups can also be used for back-referencing (matching the same pattern previously found)

```
"I kicked the the ball" %>% str_extract_all("(\\w+) \\1")
```

```
[[1]]
[1] "the the"
```

```
"I kicked the the ball" %>% str_replace_all("(\\w+) \\1", "\\1")
```

```
[1] "I kicked the ball"
```