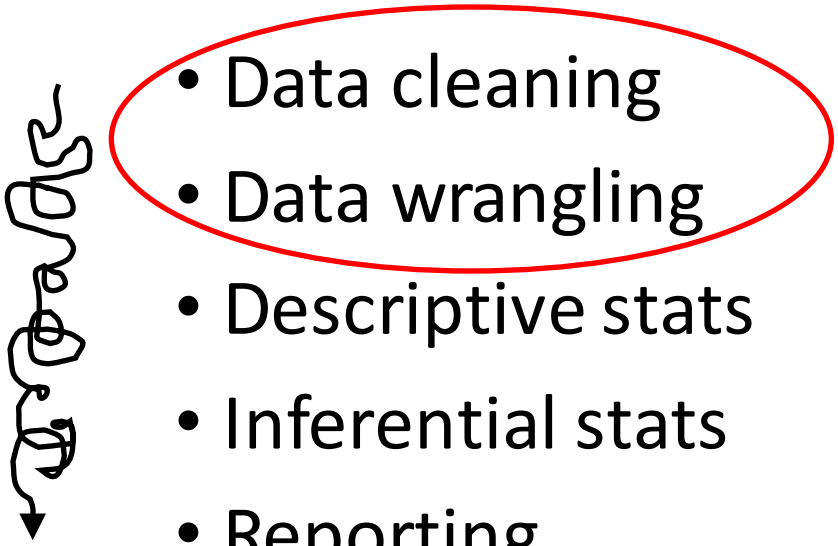
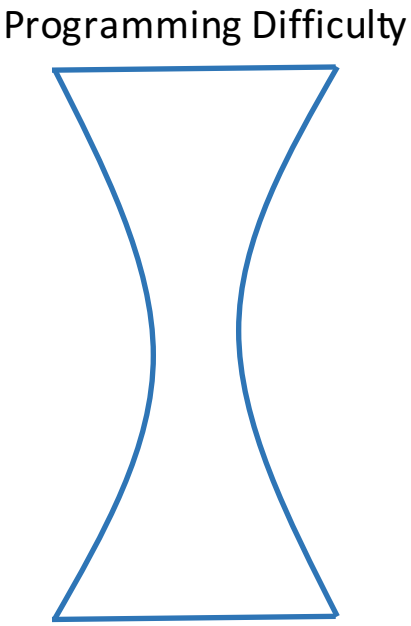
String Matching and Modification in R using Regular Expressions

Jodie Reed

The data analysis journey

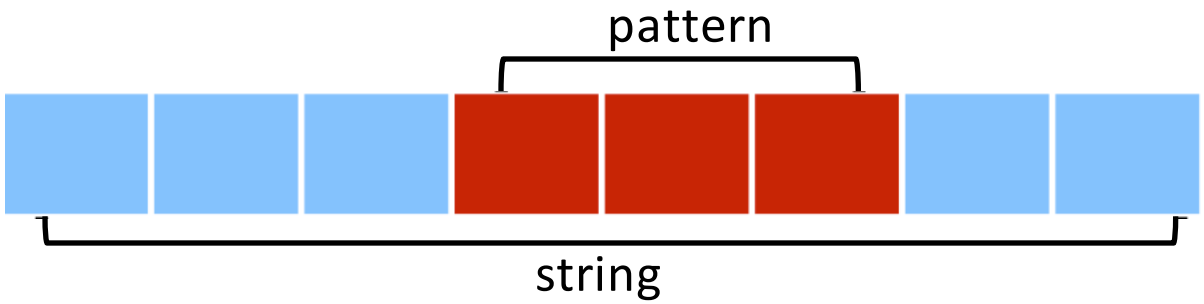




Can take up to 60-80% of data analysts time

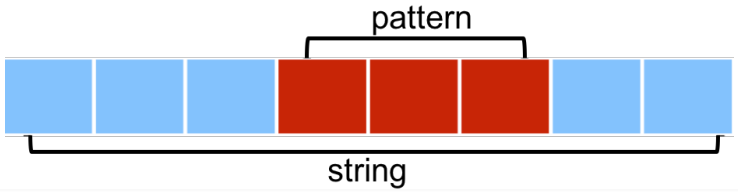
Regular Expression

- Regular expression is a pattern that describes a specific set of strings with a common structure
- Usually this pattern is then used by string searching algorithms for “find” or “find and replace” operations on strings



pattern	string
“ro”	“The frog jumped in the water.”

Functions for pattern matching: Detecting Patterns




```
> string <- c("Hiphopopotamus", "Rhymenoceros", "time for bottomless lyrics")
> pattern <- "t.m"
```

grep(pattern, string)

```
[1] 1 3
```

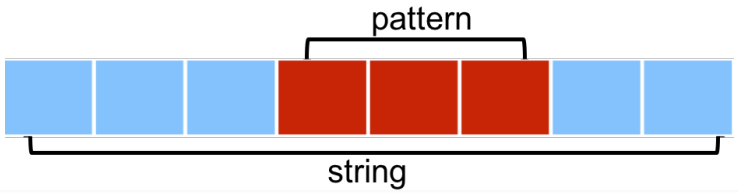
grep(pattern, string, value = TRUE)

```
[1] "Hiphopopotamus"
[2] "time for bottomless lyrics“
```

grepl(pattern, string)

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

Functions for pattern matching: Locating Patterns




```
> string <- c("Hiphopopotamus", "Rhymenoceros", "time for bottomless lyrics")
> pattern <- "t.m"
```

regexpr(pattern, string)
find starting position and length of first match

gregexpr(pattern, string)
find starting position and length of all matches

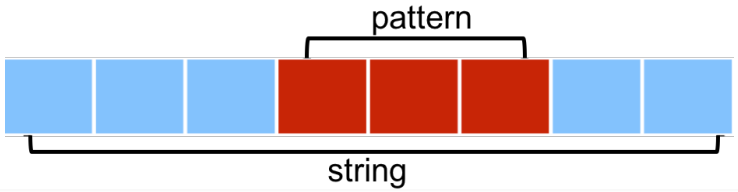
```
> regexpr(pattern, string)
[1] 10 -1  1
attr(,"match.length")
[1]  3 -1  3
attr(,"useBytes")
[1] TRUE
```

```
> gregexpr(pattern, string)
[[1]]
[1] 10
attr(,"match.length")
[1] 3
attr(,"useBytes")
[1] TRUE

[[2]]
[1] -1
attr(,"match.length")
[1] -1
attr(,"useBytes")
[1] TRUE

[[3]]
[1]  1 13
attr(,"match.length")
[1] 3 3
attr(,"useBytes")
[1] TRUE
```

Functions for pattern matching: Extracting Patterns

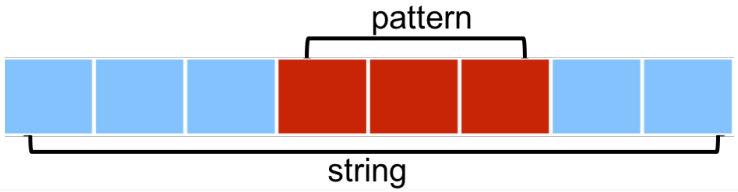



```
> string <- c("Hiphopopotamus", "Rhymenoceros", "time for bottomless lyrics")
> pattern <- "t.m"
```

regmatches(string, regexpr(pattern, string))
extract first match `[1] "tam" "tim"`

regmatches(string, gregexpr(pattern, string))
extracts all matches, outputs a list
`[[1]] "tam" [[2]] character(0) [[3]] "tim" "tom"`

Functions for pattern matching: Replacing Patterns




```
> string <- c("Hiphopopotamus", "Rhymenoceros", "time for bottomless lyrics")
> pattern <- "t.m"
```

sub(pattern, replacement, string)
replace first match

gsub(pattern, replacement, string)
replace all matches

```
> sub(pattern, "SUB", string)
[1] "HiphopopoSUBus"            "Rhymenoceros"            "SUBe for bottomless lyrics"

> gsub(pattern, "SUB", string)
[1] "HiphopopoSUBus"            "Rhymenoceros"            "SUBe for botSUBless lyrics"
```

Defining patterns:

Regular Expression Syntax:

- Regular expressions typically specify characters (or character classes) to seek out, possibly with information about repeats and location within the string
- This is accomplished with the help of metacharacters that have specific meanings: `$ * + . ? { } ^ [ ] | ( ) \`

Defining patterns:

Character Classes and Groups

.	Any character except \n
|	Or, e.g. (a|b)
[…]	List permitted characters, e.g. [abc]
[a-z]	Specify character ranges
[^…]	List excluded characters
(…)	Grouping, enables back referencing using \\N where N is an integer

Character Classes

[[:digit:]] or \\d	Digits; [0-9]
\\D	Non-digits; [^0-9]
[[:lower:]]	Lower-case letters; [a-z]
[[:upper:]]	Upper-case letters; [A-Z]
[[:alpha:]]	Alphabetic characters; [A-z]
[[:alnum:]]	Alphanumeric characters [A-z0-9]
\\w	Word characters; [A-z0-9_]
\\W	Non-word characters
[[:xdigit:]] or \\x	Hexadec. digits; [0-9A-Fa-f]
[[:blank:]]	Space and tab
[[:space:]] or \\s	Space, tab, vertical tab, newline, form feed, carriage return
\\S	Not space; [^[:space:]]
[[:punct:]]	Punctuation characters; !"#$%&'()*+,-./:;<=>?@[]^_`{|}~
[[:graph:]]	Graphical char.; [[:alnum:][:punct:]]
[[:print:]]	Printable characters; [[:alnum:][:punct:]\\s]
[[:cntrl:]] or \\c	Control characters; \n, \r etc.

```
> grep("po|hy", string, value = TRUE)
[1] "Hiphopopotamus" "Rhymenoceros"
```

```
> grep("o[pm]", string, value = TRUE)
[1] "Hiphopopotamus"              "time for bottomless lyrics"
```

```
> grep("[[:blank:]]", string, value = TRUE)
[1] "time for bottomless lyrics"
```

Defining patterns:

Anchors	
^	Start of the string
$	End of the string
\\b	Empty string at either edge of a word
\\B	NOT the edge of a word
\\<	Beginning of a word
\\>	End of a word

```
> grep("^[RrTt]", string, value = TRUE)
[1] "Rhymenoceros"                "time for bottomless lyrics"
```

```
> grep("os|cs$", string, value = TRUE)
[1] "Rhymenoceros"                "time for bottomless lyrics"
```

Defining patterns:

Quantifiers	
*	Matches at least 0 times
+	Matches at least 1 time
?	Matches at most 1 time; optional string
{n}	Matches exactly n times
{n,}	Matches at least n times
{,n}	Matches at most n times
{n,m}	Matches between n and m times

```
> grep("t{2}", string, value = TRUE)
[1] "time for bottomless lyrics"

> grep("t.{3}s", string, value = TRUE)
[1] "Hiphopopotamus"

> grep("t.*s", string, value = TRUE)
[1] "Hiphopopotamus"              "time for bottomless lyrics"
```

Defining patterns:

Escape sequences:

- Metacharacters (. * + ? etc.) cannot be directly coded in a string
- Need to ”escape” the character by preceding with \\

- Search using metacharacter “*” (returns all matches)

```
> grep("*", string, value = TRUE)
[1] "Hiphopopotamus"              "Rhymenoceros"              "time for bottomless lyrics"
```

- Search for a literal “*” by escaping (returns no matches)

```
> grep("\\*", string, value = TRUE)
character(0)
```

Defining patterns:

Back referencing:

- (…) : grouping
- This allows you to retrieve bits that matched various parts of your regular expression so you can alter them or use them for building a new string
- Group can then be referred to using \\N with N being the number of (…) used
- I.O.W. search for a pattern, alter part of pattern not in group

- Here we don't use grouping so the entire pattern is replaced

```
> sub("popo", "OPOP", string)
[1] "HiphoOPOPtamus"              "Rhymenoceros"              "time for bottomless lyrics"
```

- Here we use grouping and refer to group in the replacement

```
> sub("(pop)o", "\\1OPOP", string)
[1] "HiphopopOPOPtamus"           "Rhymenoceros"              "time for bottomless lyrics"
```

Exercise:

- Using gapminder data: data on life expectancy, GDP per capita, and population per country.
- In RStudio run the following commands:

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

- Check the structure of the dataset

```
str(gapminder)
```

- Create an object of levels of Countries from gapminder and label it “string”

```
string <- levels(gapminder$country)
```

- Check the structure of the object “string”

```
str(string)
```

Exercise:

Use the cheatsheet found at https://www.rstudio.com/wp-content/uploads/2016/09/RegExCheatsheet.pdf as a reference

1. Find all countries in gapminder with names that contain the letter ‘w’ (or ‘W’)

   (hint: use grep(‘pattern’, string, value=T))
2. Find all countries with names that contain the letter ‘w’ or ‘z’ (only lower case)
3. Find all countries with names that contain the pattern ‘af’ anywhere in the string
4. Find all countries with names that begin with ‘Af’
5. Find all countries with names that end with ‘land’

Exercise:

6. Find all countries with names that contain ‘i’ or ‘t’ and end with ‘land’
7. Find all countries with names that contain a full stop
8. Find all countries with names that contain any punctuation
9. Find all countries with names that contain ’ee’
10. Find countries with names that contain ‘i’ or ‘t’ and end with ‘land’, and replace ’land’ with ’LAND’ using back referencing (hint: for this we would use the sub() function)

Answers to exercise:

```
#1. Find all countries in gapminder with names that contain the letter ‘w’
#   (or 'W')
grep("[Ww]", string, value=TRUE)


#2. Find all countries  with names that contain the letter ‘w’ or ‘z’
grep("w|z", string, value=TRUE)


#3. Find all countries with names that contain the pattern ‘af' anywhere
#   the string
grep("[Aa]f", string, value=TRUE)


#4. Find all countries with names that begin with ‘Af’
grep("^[Aa]f", string, value=TRUE)


#5. Find all countries with names that end with ‘land’
grep("land$", string, value=TRUE)
```

Answers to exercise:

```r
#6. Find all countries with names that contain ‘i’ or ‘t’ and end with ‘land’
#   (hint: search for uppercase and lowercase 'i' or 't')
grep("[IiTt].*land$", string, value=TRUE)


#7. Find all countries with names that contain a full stop
grep("\\.", string, value=TRUE)


#8. Find all countries with names that contain any punctuation
grep("[[:punct:]]", string, value=TRUE)


#9. Find all countries with names that contain ’ee’
#   (Hint: use the quantifiers in this one)
grep("e{2}", string, value = TRUE)


#10. Find countries with names that contain ‘i’ or ‘t’ and end with ‘land’,
#    and replace ’land’ with ’LAND’ using back referencing
#    (hint: for this we would use the sub() function)
sub("([IiTt].*)land$", "\\1LAND", string)
#to check
grep("LAND$", sub("([IiTt].*)land$", "\\1LAND", string), value=TRUE)
```