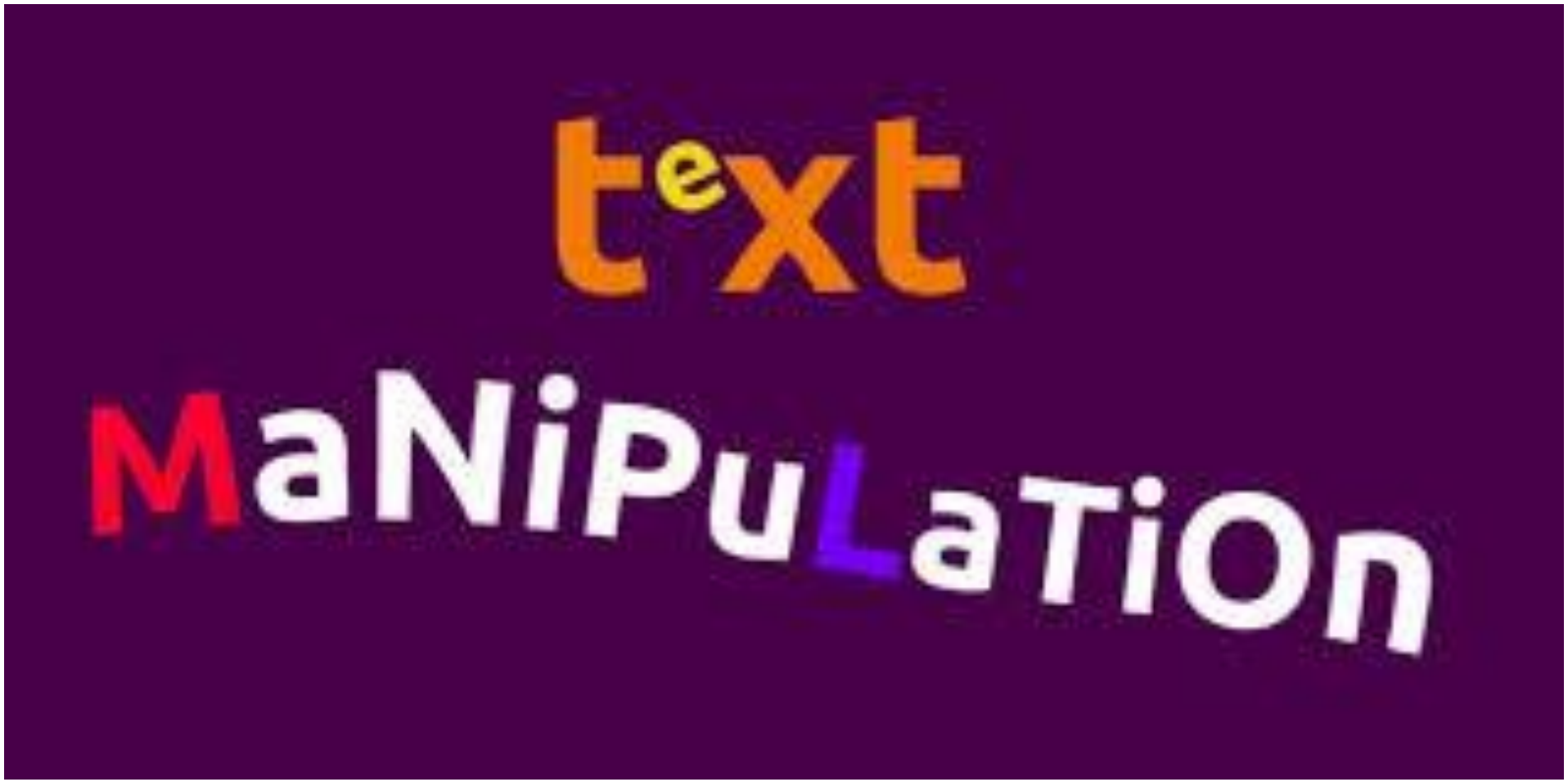

text
MaNiPuLaTiOn

Overview

String manipulation with stringr

Regular expressions

Exercises using regular expressions

Text manipulation

80% of a Data Scientists time is cleaning data

- Text manipulation is a big part of cleaning data

20% of a Data Scientists time is complaining about cleaning data

Text manipulation in R

Base R has a number of text manipulation functions

```
> tolower("Hey")
```

stringr is the tidyverse version

- Does many of the same things with a more consistent syntax (e.g., all functions start with str_)

```
> str_to_lower("STOP YELLING")
```

We will focus on stringr because it’s a bit of an improvement over base R's functions

```
> library(stringr)
```

str_trim and str_pad

str_trim removes leading and trailing whitespace.

- Similar to base R: strtrim()

Example:

```
> str_trim("    What a mess      ")
```

str_pad adds extra whitespace whitespace

Example:

```
> str_pad("Let's make it messier", 50, "right")
> str_pad(1:11, 3, pad = 0)   # useful for adding leading 0's
```

str_sub

Returns a substring from the original string

- str_sub("String", start.m, end.n)
- Equivalent to base R: substr()

Examples:

```
> str_sub("What a mess", 6, 11)

> fruits <- c("apple", "pineapple", "Pear", "orange",  "peach", "banana")
> str_sub(fruits, 2, 4)
```

str_c

Combines a number of strings together

- Equivalent to base R: paste()

Examples:

Make sure there is a space here

```
> str_c("What", "a", "mess", sep = " ")

# we can also concatenate values in a vector
> vec_words <- c("What", "a", "mess")
> str_c(vec_words, collapse = " ")
```

Let's download a web page

```
>  base_name <- " https://www.nytimes.com/2021/11/17/us/

>  article_name <- " catholic-bishops-biden-communion.html"

> full_name <- str_c(base_name, article_name)

> download.file(full_name, article_name)

> viewer <- getOption("viewer")

> viewer(article_name)
```

str_length

Returns the number of characters in the string

- Equivalent to base R: nchar()

Examples:

```
> str_length("What a mess")

> article_size <- file.info(article_name)$size    # size of the article in bytes

# read the whole article as a single string
> the_article <- readChar(article_name, article_size)

> str_length(the_article)    # size of the article as a string
```

str_replace_all

Takes a string, and replaces every instance of substring with a new string

- > str_replace("String", "old", "new")
- Equivalent to base R: gsub()

Example:

```
> article2  <- str_replace_all(the_article,  "Biden", "Sleepy Joe")

> write(article2, "sleepy_article.html")

> viewer("sleepy_article.html")
```

str_split

Splits a single string into a list of strings

- > str_split("String", "split pattern")
- Equivalent to base R: strsplit()

Make sure there is a space here

Examples:

```
> list_of_strings <- str_split("What a mess", " ")
> vector_of_strings <- unlist(list_of_strings)
> vector_of_strings[3]


> article_vec <- unlist(str_split(the_article, " "))
```

str_extract

Extract a pattern from a string

- > str_extract("String", "pattern")
- Equivalent to base R: sub()

Examples:

> str_extract(fruits, "apple")

str_detect

Check to see if a pattern occurs in a string

- > str_detect("String", "pattern")
- Equivalent to base R: grepl()

Examples:

```
> str_detect(fruits,  "apple")

# can you tell how many times Biden was mentioned in the article?
> sum(str_detect(article_vec, ”Biden"))
```

Regular expressions

Regular expressions are string that allow you find more complex patterns in strings

For example:

- The character "^" indicates the beginning of a string
- The character "$" indicates the end of a string
- The expression "[Pp]" indicates "P" or "p"

```
# what do these expressions do?
> str_detect(fruits, "e$")
> str_detect(fruits, "^[Pp]")
```

The following are special regular expression characters that are reserved:

```
.   *   \   $   {}   []    ^    ?
```

Regular expressions

. (period) matches any single character
- > str_detect(c("mess", "mass", "miss"), "m.ss")

* means match 0 or more of the preceding character
- > str_detect(c("xz", "xyz", "xyyz", "xyyyz"), "xy*z")

+ means match 1 or more of the preceding character
- > str_detect(c("xz", "xyz", "xyyz", "xyyyz"), "xy+z")

what will the following match?
- > str_detect(fruits, "^a.*e$")

what about if the ^ was removed?

Regular expressions

[ ] means match anything in the range inside the braces

- > str_detect(fruits, "^[a-o]")
- > str_detect(c("chimp", "champion", "chomp"), "ch[aio]mp")

Note: if the ^ appears inside square braces it means not

- > str_detect(fruits, "^[^a-o]")

() groups things together, useful in combination with {}

{num} means repeat the preceding sequence num times

- > str_detect(fruits, "(an){2}")
- > str_extract(fruits, "(an){1,}")

Example

```
strings  <-  c(
        "apple",
        "219  733  8965",
        "329-293-8753",
        "Work: 579-499-7527",
        "Home: 543.355.3679"
)

phone <-  "([2-9][0-9]{2})[- .]([0-9]{3})[- .]([0-9]{4})"

str_extract(strings, phone)
```

Escape sequences

In regular expressions a period (.) means any character

- So how can you detect if a period is in a string?

Escape sequences in R start with two slashes \\ and cause the next character to be treated literally rather than as a special character

- To match a period we use \\. [.] also works
- To match a $ symbol we use \\$

Extract the amounts of money and dollar sign from this string (use str_extract_all)

- > the_string <- c("Sasha has $100 and Harry has $0")
- > str_extract_all(the_string, "\\$[0-9]{1,}")

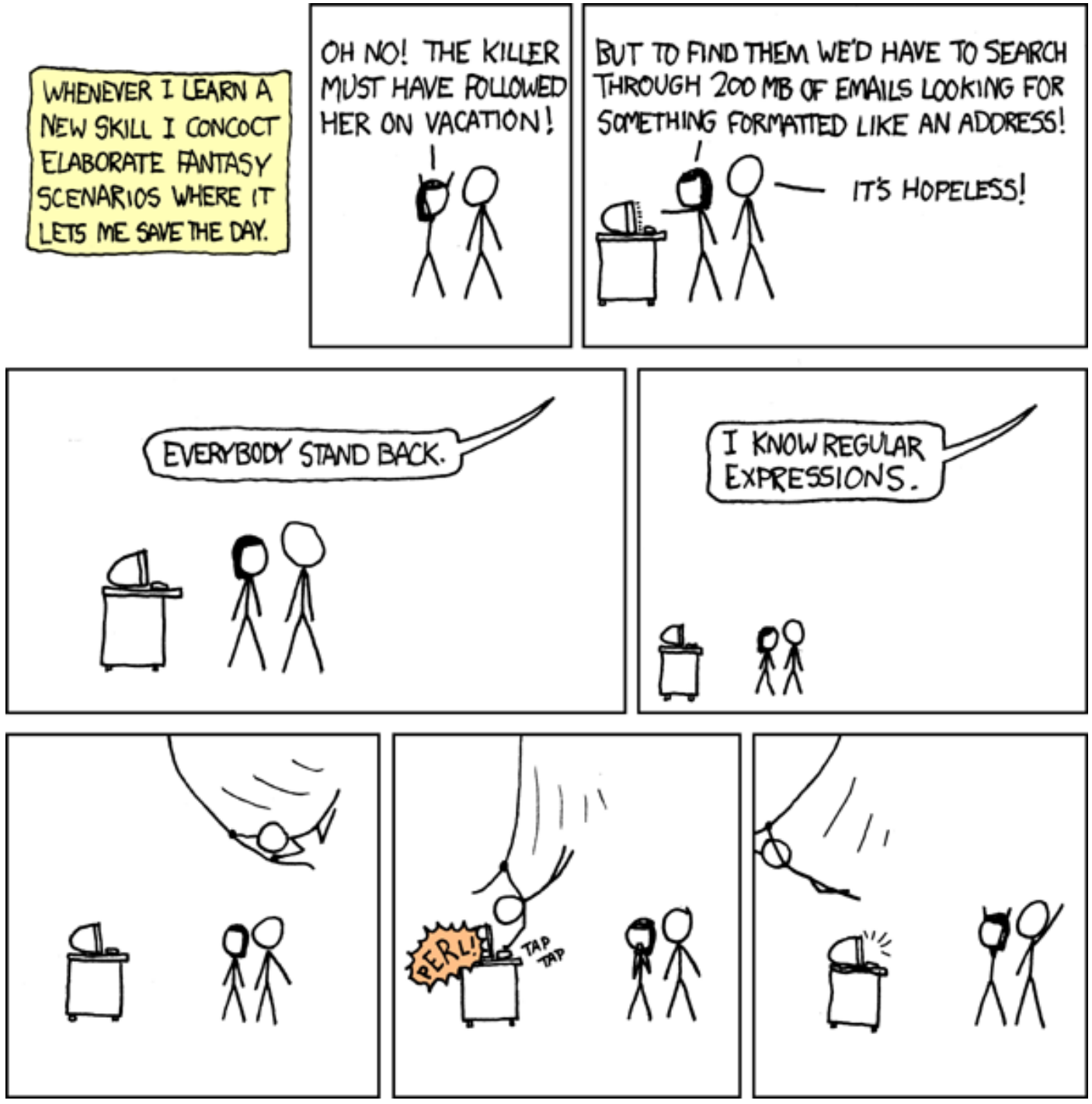
Character classes

Other special characters are also designated by using a double slash first

- \\s space
- \\n new line or also \\r
- \\t tab

```
# get 6 characters prior to the end of a line in the_article
>  str_extract_all(the_article, ".{6}\\n")


# all ending html tags
> end_tag <- str_extract_all(the_article, "</[A-z]{1,}>\\n")
> lapply(end_tag, str_replace, "\n", "")
```


WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

Pig Latin

Pig Latin is language game that conceals words for others not familiar with the rules

The suffix "way" is added to the end of words that start with vowels

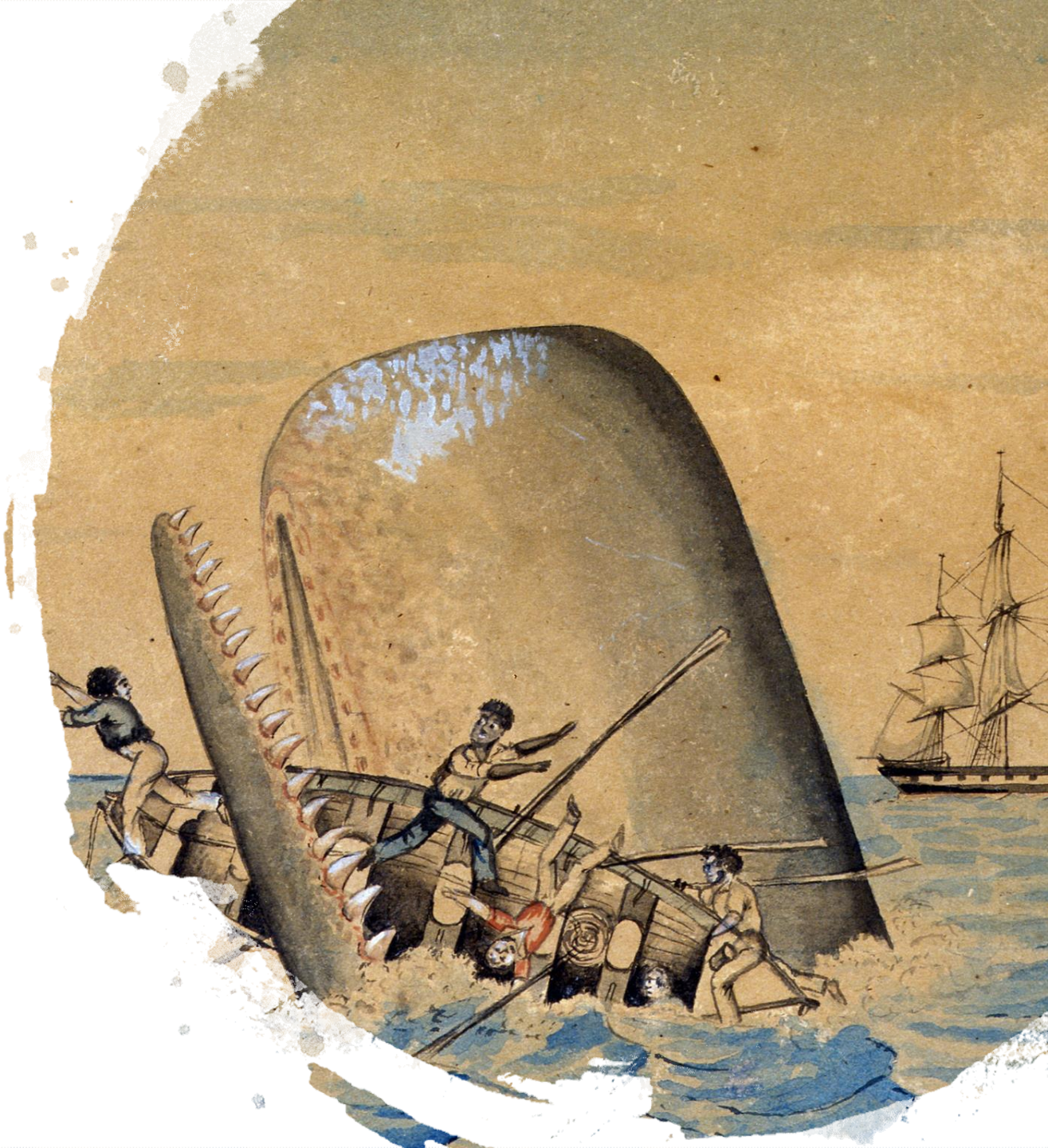
- E.g., eat -> eatway, are -> areway

For words that start with consonants, the first letter is removed from the start of the word and added to the end of the word with an additional "ay"

- E.g. pig -> igpay latin -> atinlay

Moby Dick in Pig Latin

To practice using regular expressions and the stringr package, let's convert the Moby Dick back into its original Pig Latin



Text analysis

Specific word choices can reveal who wrote particular documents

Computer algorithms have been developed to detect specific authors

- Multiple authors wrote different sections of the bible
- Predicting who wrote unsigned court opinions

One can also look at region dialects on Twitter

- Work done by Brendan O'Connor at UMass