STAT570-HW2

Levent Sarı - 2290930

Pattern Control

It's possible to exercise extra control over the details of the match by using a pattern object instead of just a string. This allows you to control the so called regex flags and match various types of fixed strings, as described below.

15.5.1 Regex flags

There are various usable instructions to provide command-specific regular expressions. These instructions are generally denoted to be "flags". Using stringr, it is possible to use those commands inside the `regex()` command. First instruction we will focus on will be the `ignore_case`. Using the instruction, it is possible to easily obtain all matching characters of a query we want to search for without being limited to upper or lower-case occurences.

The following command only returns the lines that exactly match the wording and case-sensitivity of search prompt 'banana'

```
library(tidyverse)
library(stringr)
```

```
bananas <- c("banana", "Banana", "BANANA")
str_view(bananas, "banana")
```

```
[1] | <banana>
```

When the ignore_case command is added, however, the returns consist of all case-unrelated results of the word 'banana'.

```
str_view(bananas, regex("banana", ignore_case = TRUE))
```

```
[1] | <banana>
[2] | <Banana>
[3] | <BANANA>
```

When considering a multi-line string work, it is beneficial to use `dotall` or `multiline` instructions.

- When `dotall = TRUE`, the `.` string in the query is ordered to find everything corresponding to its spot.

```
x <- "Line 1\nLine 2\nLine 3"
str_view(x, ".Line")
str_view(x, regex(".Line", dotall = TRUE))
```

```
[1] | Line 1<
    | Line> 2<
    | Line> 3
```

```
#> [1]   Line 1<
#>       Line> 2<
#>       Line> 3
```

The command above provides the locations of every pre-'Line' word by finding the '\n' operators before 'Line' commands, as `dotall = TRUE` instructs it to.

- `multiline = TRUE` makes `^` and `$` match the start and end of each line rather than the start and end of the complete string:

```
x <- "Line 1\nLine 2\nLine 3"
str_view(x, "^Line")
```

```
[1] | <Line> 1
    | Line 2
    | Line 3
```

```
#> [1]   <Line> 1
#>       Line 2
#>       Line 3
str_view(x, regex("^Line", multiline = TRUE))
```

```
[1] | <Line> 1
    | <Line> 2
    | <Line> 3
```

```
#> [1]   <Line> 1
#>       <Line> 2
#>       <Line> 3
```

It is also possible to commentate on regular expressions similar to base R using `#`. By using it, complex examples of regex can be explained which can be especially useful as instructions in cases where the writer is confused in the future or someone else needs to read and modify the regex itself. It is possible to use white-spaces and every operator after the `#` marker. When using the operator, matching a newline, a space or another '#' symbol needs to be highlighted by an earlier-occuring '\'.

```
phone <- regex(
  r"(
    \(?     # optional opening parens
    (\d{3}) # area code
    [)\-]?  # optional closing parens or dash
    \ ?     # optional space
    (\d{3}) # another three numbers
    [\ -]?  # optional space or dash
    (\d{4}) # four more numbers
  )",
  comments = TRUE
)

str_extract(c("514-791-8141", "(123) 456 7890", "123456"), phone)
```

```
[1] "514-791-8141"   "(123) 456 7890" NA
```

```
#> [1] "514-791-8141"   "(123) 456 7890" NA
```

15.5.2 Fixed matches

Using `fixed()` lets one bypass the regex rules:

```
str_view(c("", "a", "."), fixed("."))
```

```
[3] | <.>
```

```
#> [3]   <.>
```

Even though `fixed()` bypasses the regex instructions, it is still possible to overcome case-sensitive problems using the same instruction:

```
str_view("x X", "X")
```

```
[1] | x <X>
```

```
#> [1]   x <X>
str_view("x X", fixed("X", ignore_case = TRUE))
```

```
[1] | <x> <X>
```

```
#> [1]   <x> <X>
```

Using regex on languages other than English, usage of `coll()` is recommended versus using `fixed()`, as it can read the string according to the specified `locale` .

```
str_view("i İ ı I", fixed("İ", ignore_case = TRUE))
```

```
[1] | i <İ> ı I
```

```
#> [1]   i <İ> ı I
str_view("i İ ı I", coll("İ", ignore_case = TRUE, locale = "tr"))
```

```
[1] | <i> <İ> ı I
```

```
#> [1]   <i> <İ> ı I
```

15.5.L Self-made Example

Künstliche Intelligenz, auch bekannt als \nKi, ûnd ihr ûmfassender Einfluss in unserer modernen\nGesellschaft sind ûnleugbar. Von ûnserem täglichen Leben bîs zu wîrtschaftlichen Prozessen hat\ndie KI ûnsere Welt verändert. Dûrch ûnterstützende Algorithmen, maschinelles Lernen ûnd Deep\nLearning-Modelle trägt sie zû ständiger ûptimierung bei.Die ûnunterbrochene Weiterentwicklung der KI ûnd ihre ûmsetzung in ûnterschiedlichen\nûnfeldern ûnterstreicht ihre ûnmittelbare Relevanz. Ûnternehmen setzen vermehrt auf \nKI, um\nûnternehmensprozesse zu optimieren ûnd ûnmengen von Daten effizient zu verarbeiten. Ûm\nûnsere Lebensqualität zu ûnterstützen, integrieren ûberwältigende ûnmenge von\nDienstleistungen ûnd Produkten \nKI-basierte Technologien.\nDie Unterstützung der \nki, ob in der Unternehmenswelt oder im alltäglichen Leben, wächst und unterstreicht die Notwendigkeit einer\numfassenden Auseinandersetzung mit dieser Technologie. Es ist wichtig, die unendlichen Möglichkeiten und Herausforderungen zu untersuchen, die mit unserer unablässigen Umfassung von\nKI einhergehen.

```
my_sentence = 'Künstliche Intelligenz, auch bekannt als \nKi, ûnd ihr ûmfassender Einfluss
```

Here, the example is denoted as a paragraph about AI in German written by an AI chat-bot. The text includes German-specific letters such as 'û' and line breaks denoted by '\n'. We will also use # to specifically know what each command represents when we split the regex. We will now separately find all occurences of the word 'ki' as well as 'û'.

- Finding all occurences of 'ki' that occur at the beginning of lines.

Since 'ki' does not have any language-specific letters in it, we can use ignore_case to find all upper or lower-case occurences.

```
library(stringr)
str_view(my_sentence, regex('ki', ignore_case = TRUE))
```

```
[1] | Künstliche Intelligenz, auch bekannt als
    | <Ki>, ûnd ihr ûmfassender Einfluss in unserer modernen
    | Gesellschaft sind ûnleugbar. Von ûnserem täglichen Leben bîs zu wîrtschaftlichen Proze
    | die <KI> ûnsere Welt verändert. Dûrch ûnterstützende Algorithmen, maschinelles Lernen
    | Learning-Modelle trägt sie zû ständiger ûptimierung bei.Die ûnunterbrochene Weiterentw
    | ûnfeldern ûnterstreicht ihre ûnmittelbare Relevanz. Ûnternehmen setzen vermehrt auf
    | <KI>, um
    | ûnternehmensprozesse zu optimieren ûnd ûnmengen von Daten effizient zu verarbeiten. Ûm
    | ûnsere Lebensqualität zu ûnterstützen, integrieren ûberwältigende ûnmenge von
    | Dienstleistungen ûnd Produkten
    | <KI>-basierte Technologien.
```

```
    | Die Unterstützung der
    | <ki>, ob in der Unternehmenswelt oder im alltäglichen Leben, wächst und unterstreicht d
    | umfassenden Auseinandersetzung mit dieser Technologie. Es ist wichtig, die unendlichen
    | <KI> einhergehen.
```

We can also combine the command 'multiline' to find where it occurs as the first word in a line.

```
library(stringr)
str_view(my_sentence, regex('^ki', multiline = TRUE, ignore_case = TRUE))
```

```
[1] | Künstliche Intelligenz, auch bekannt als
    | <Ki>, ûnd ihr ûmfassender Einfluss in unserer modernen
    | Gesellschaft sind ûnleugbar. Von ûnserem täglichen Leben bîs zu wîrtschaftlichen Proze
    | die KI ûnsere Welt verändert. Dûrch ûnterstützende Algorithmen, maschinelles Lernen ûn
    | Learning-Modelle trägt sie zû ständiger ûptimierung bei.Die ûnunterbrochene Weiterentw
    | ûnfeldern ûnterstreicht ihre ûnmittelbare Relevanz. Ûnternehmen setzen vermehrt auf
    | <KI>, um
    | ûnternehmensprozesse zu optimieren ûnd ûnmengen von Daten effizient zu verarbeiten. Ûm
    | ûnsere Lebensqualität zu ûnterstützen, integrieren überwältigende ûnmenge von
    | Dienstleistungen ûnd Produkten
    | <KI>-basierte Technologien.
    | Die Unterstützung der
    | <ki>, ob in der Unternehmenswelt oder im alltäglichen Leben, wächst und unterstreicht d
    | umfassenden Auseinandersetzung mit dieser Technologie. Es ist wichtig, die unendlichen
    | <KI> einhergehen.
```

- Finding all occurences of 'û' that occur at the beginning of lines.

Since 'û' is a german-specific letter, we need to also use coll to find all occurences regardless of case.

```
library(stringr)
str_view(my_sentence, coll('û', ignore_case = TRUE, locale = 'de'))
```

```
[1] | Künstliche Intelligenz, auch bekannt als
    | Ki, <û>nd ihr <û>mfassender Einfluss in unserer modernen
    | Gesellschaft sind <û>nleugbar. Von <û>nserem täglichen Leben bîs zu wîrtschaftlichen Pr
    | die KI <û>nsere Welt verändert. D<û>rch <û>nterstützende Algorithmen, maschinelles Lern
    | Learning-Modelle trägt sie z<û> ständiger <û>ptimierung bei.Die <û>nunterbrochene Weite
    | <û>nfeldern <û>nterstreicht ihre <û>nmittelbare Relevanz. <Û>nternehmen setzen vermehrt
```

```
| KI, um
| <û>nternehmensprozesse zu optimieren <û>nd <û>nmengen von Daten effizient zu verarbeit
| <û>nsere Lebensqualität zu <û>nterstützen, integrieren <û>berwältigende <û>nmenge von
| Dienstleistungen <û>nd Produkten
| KI-basierte Technologien.
| Die Unterstützung der
| ki, ob in der Unternehmenswelt oder im alltäglichen Leben, wächst und unterstreicht di
| umfassenden Auseinandersetzung mit dieser Technologie. Es ist wichtig, die unendlichen
| KI einhergehen.
```

Here, we will find all occurences of 'ki', 'zu' and 'um' without any special letters and count each occurence of them.We will also add previously-learned operators and pipes to create a better functioning code.

```
pattern <- regex(
  r"(
    ki|   # Matches 'ki'
    zu|   # Matches 'zu'
    um    # Matches 'um'
  )",
  ignore_case = TRUE,
  comments = TRUE,
  multiline = TRUE
)

my_sentence %>%
  str_extract_all(pattern) %>%
  lapply(tolower) %>%
  unlist() %>%
  table() %>%
  print()
```

```
.
ki um zu
 7  3  8
```