데이터 분석의 기본이 되는 데이터전처리

Step2. 데이터 전처리에 유용한 정규표현식과 유용한 함수 학습하기

https://mrchypark.github.io/dabrp_classnote3/class4

박찬엽

2017년 10월 12일

목차

과제 확인

예시 코드

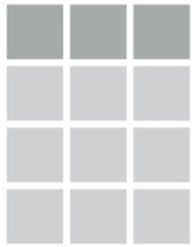
수업의 목표

1. 정규표현식의 문법을 이해하고 동작을 예상할 수 있다.

2. 많이 사용하는 정규표현식을 이해하고 응용할 수 있다.

3. stringr 패키지와 함께 전처리에 정규표현식을 사용할 수 있다.

데이터 다루기



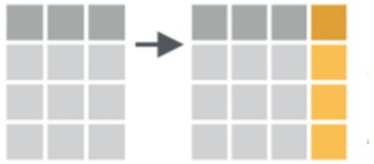
열 방향

행 방향

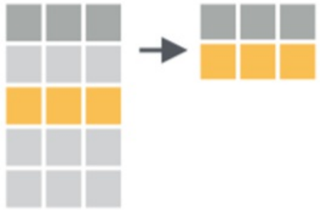
선택

① 

계산

② 

조건

③ 

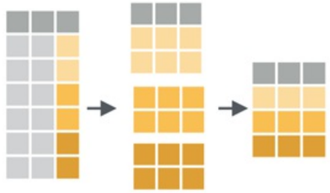
추가

④ 

정렬

⑤ 

그룹 계산

⑥ 

열 결합

⑦

x

A	B	C
a	t	1
b	u	2
c	v	3

+

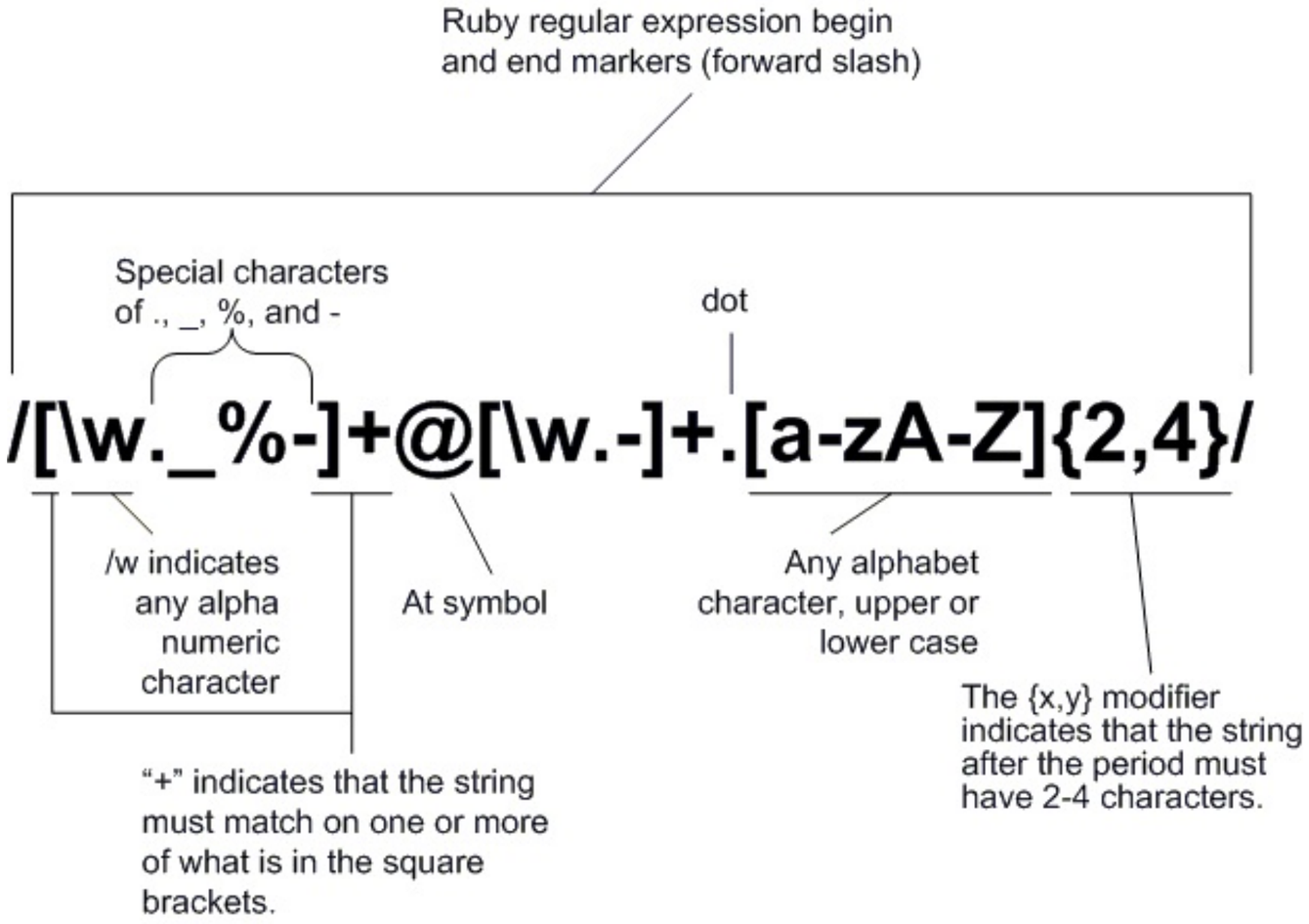
y

A	B	D
a	t	3
b	u	2
d	w	1

=

A	B	C	D
a	t	1	3
b	u	2	2
c	v	3	NA

정규표현식이란



username @ domain . qualifier (com/net/tv/…)

정규표현식이란

특정한 규칙을 가진 문자열의 집합을 표현하는데 사용하는 형식 언어

-> 문법을 외워야 해서 읽고 사용하기 어려움

-> 익숙해지면 글자를 다루는 코드를 잘 작성할 수 있음

R의 정규표현식

콘솔에 `?regex`를 입력해 보세요

- 요약
  - 표준문법인 POSIX와 perl방식의 PCRE 2가지가 대표적이며
  - R은 POSIX의 basic과 Extended 중 Extended를 지원
  - `perl = T` 옵션으로 PCRE 방식을 사용할 수도 있음

유용한 페이지

정규표현식 설명

정규표현식 시각화

R for data science#strings

정규표현식의 R에서 사용(한글)

메타문자

- ^ : 문자열의 시작
- $: 문자열의 종료
- . : 어떤 문자던 한 개를 표현
- []: 문자클래스로서 괄호안에 조건에 해당하는 글자 한 개를 표현
- ? : 앞에 있는 문자가 없거나 하나
- + : 앞에 있는 문자가 하나 이상
- * : 앞에 있는 문자가 없거나 하나 이상
- {n,m} : 앞 문자가 n개 이상 m개 이하
- () : 하나의 그룹으로 지정하고 글자처럼 취급하며 이후에 ₩1 등으로 사용 가능

단순 매칭

grep은 찾고자하는 문자열이 있는지 찾아주는 함수입니다. 단순 매칭의 경우 찾고자하는 패턴에 character 데이터를 입력해주면 같은 글자를 가진 vector의 위치를 반환합니다.

grep(찾고자하는 패턴, 대상벡터)

?grep를 실행해서 확인해 보세요.

```
data<-c("apple","banana","banano")
grep("banana", data)
```

```
## [1] 2
```

```
grepl("banana", data)
```

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

문자열의 시작

단순 매칭하는 상황에서 "안에 패턴을 작성할 때 ^을 맨 앞에 같이 사용하면 그 뒤의 글자로 시작하는 데이터만 찾습니다.

```
data<-c("apple","banana","banano", "a banana")
grep("banana", data)
```

```
## [1] 2 4
```

```
grep("^banana", data)
```

```
## [1] 2
```

문자열의 끝

단순 매칭하는 상황에서 "안에 패턴을 작성할 때 $을 맨 뒤에 같이 사용하면 그 앞의 글자로 끝나는 데이터만 찾습니다.

```
data<-c("apple","banana","banano", "a banana", "a banana a")
grep("banana", data)
```

```
## [1] 2 4 5
```

```
grep("banana$", data)
```

```
## [1] 2 4
```

완전히 일치하는 경우만

문자열의 시작과 끝을 강제하는 방법을 배웠으니 완전히 일치하는 방법을 사용할 수 있게 되었습니다.

```
data<-c("apple","banana","banano", "a banana", "a banana a")
grep("banana", data)
```

```
## [1] 2 4 5
```

```
grep("^banana$", data)
```

```
## [1] 2
```

사용해 보기

- nycflights13 패키지의 airports 데이터에 이름에 New가 포함되는 데이터는 몇 개 인가?
- nycflights13 패키지의 airports 데이터에 이름이 New로 시작하는 데이터는 몇 개 인가?

```
if (!require(nycflights13)) install.packages("nycflights13")
```

```
## Loading required package: nycflights13
```

```
library(nycflights13)
head(airports, 3)
```

```
## # A tibble: 3 x 8
##     faa                           name      lat       lon   alt    tz   dst
##   <chr>                          <chr>    <dbl>     <dbl> <int> <dbl> <chr>
## 1   04G             Lansdowne Airport 41.13047 -80.61958  1044    -5     A
## 2   06A Moton Field Municipal Airport 32.46057 -85.68003   264    -6     A
## 3   06C           Schaumburg Regional 41.98934 -88.10124   801    -6     A
## # ... with 1 more variables: tzone <chr>
```

임의의 글자 한 개

`.` 은 정규표현식에서 무엇이든 한 개의 글자를 의미합니다.

```
x <- c("apple", "banana", "pear")
grep(".a.", x)
```

```
## [1] 2 3
```

메타문자를 글자그대로

\를 메타문자 앞에 쓰면 메타문자로서가 아니라 글자 그대로로 인식합니다. 그런데 \ 또한 메타문자로서 동작하기 때문에 \\를 작성해주어야 합니다.

```
x <- c("apple", "banana", "pear", ".apple")
grep("\\.a.", x)
```

```
## [1] 4
```

```
grep("\.a.", x)
```

```
Error: '\.' is an unrecognized escape in character string starting ""\."
```

문자 클래스

문자 클래스를 표현하는 []는 대괄호 안에 있는 글자 하나하나가 문자클래스로 가능한 경우입니다. 예를 들어 [02468]이라고 하면 0, 2, 4, 6, 8 중 하나의 글자면 같은 패턴으로 이해합니다.

```
x <- c("123", "1357","999990","1133")
grep("[02468]", x)
```

```
## [1] 1 3
```

문자 클래스 내에서는 ^가 지정한 글자들을 제외하고라는 뜻입니다.

```
x <- c("123", "1357","999990","0200","02468")
grep("[^02468]", x)
```

```
## [1] 1 2 3
```

문자 클래스의 연속

작성하는 문자 클래스가 범위를 가질 경우 -를 통해서 앞에 글자에서부터 뒤의 글자까지라는 의미로 사용합니다. 예를 들어 숫자 전체는 [0-9]로 표현되고 소문자 알파벳 전체라면 [a-z]라고 합니다. 알파벳 전체는 [a-zA-Z]라고 합니다. 한글도 동작해서 [ㄱ-ㅎ](자음 전체), [가-힣](한글 전체) 등등도 지원합니다.

관례적 문자 클래스

문자 클래스는 관례적으로 여러 단축 표현이 있습니다. `[[:xxxx:]]`의 형태를 띄고 의미는 아래와 같습니다.

[[:ascii:]] ASCII 문자(모두 128)
[[:alpha:]] 알파벳 문자(영문자)
[[:digit:]] 숫자
[[:alnum:]] 영문자와 숫자
[[:blank:]] 빈 문자(스페이스, 탭 등 전체)
[[:space:]] 공백 문자
[[:lower:]] 소문자
[[:upper:]] 대문자

기타 문자클래스

사용해 보기

- nycflights13 패키지의 airports 데이터에 이름이 숫자로 끝나는 데이터는 몇 개 인가? (2가지)

```
if (!require(nycflights13)) install.packages("nycflights13")
library(nycflights13)
head(airports, 3)
```

```
## # A tibble: 3 x 8
##     faa                          name      lat       lon   alt    tz   dst
##   <chr>                         <chr>    <dbl>     <dbl> <int> <dbl> <chr>
## 1   04G             Lansdowne Airport 41.13047 -80.61958  1044    -5     A
## 2   06A Moton Field Municipal Airport 32.46057 -85.68003   264    -6     A
## 3   06C           Schaumburg Regional 41.98934 -88.10124   801    -6     A
## # ... with 1 more variables: tzone <chr>
```

앞의 글자가 없거나 하나

?는 글자 뒤에 붙어서 그 글자가 한개 있거나 없는 경우 모두를 표현할 때 사용합니다.

```
x <- c("apple", "banana", "pear", "aple")
grep("app?", x)
```

```
## [1] 1 4
```

앞의 글자가 하나 이상

+는 글자 뒤에 붙어서 그 글자가 한개 이상 연속하는 모두를 표현할 때 사용합니다.

```
x <- c("apple", "banana", "pear", "aple")
grep("p+", x)
```

```
## [1] 1 3 4
```

```
grep("ap+", x)
```

```
## [1] 1 4
```

앞의 글자가 없거나 하나 이상

*는 글자 뒤에 붙어서 그 글자가 없는 경우부터 여러 개 연속하는 모두를 표현할 때 사용합니다.

```
x <- c("apple", "banana", "pear", "aple", "abble","appppppppple")
grep("app*", x)
```

```
## [1] 1 4 6
```

글자의 갯수를 조절하기

앞의 메타 문자는 모두 없거나 하나, 아니면 갯수와 상관없이 연속되는 경우를 뜻합니다. 그래서 이제 몇 개에서 부터 몇 개까지만 연속하는 것을 찾고 싶을 때 사용하는 문법이 있습니다. `{ }`은 몇 가지 사용법이 있는데,

`{n}` : 글자가 n개인 경우
`{n, }` : 글자가 n개 이거나 더 많은 경우 `{ ,m}` : 글자가 m개 이거나 더 적은 경우 `{n,m}` : 글자가 n개에서 부터 m개 사이에 있는 경우

정말 그렇게 동작할까

a{3}은 a가 3개 연속하는 경우를 뜻합니다. 하지만 aaaa또한 3개 연속하는 경우가 그 안에 있기 때문에 조건을 만족하게 됩니다. 이렇기 때문에 항상 생각하는 대로 동작하는지 확인해야 합니다.

```
x <- c("a","aa","aaa","aaaa","aaaaa")
grep("a{3}", x)
```

```
## [1] 3 4 5
```

```
grep("^a{3}$", x)
```

```
## [1] 3
```

```
grep("a{3,}", x)
```

```
## [1] 3 4 5
```

```
grep("a{,3}", x)
```

```
## [1] 1 2 3 4 5
```

```
grep("a{2,3}", x)
```

```
## [1] 2 3 4 5
```

?를 활용한 조절

일반적으로 갯수가 더 많은 갯수의 결과를 보여주는데, 갯수를 의미하는 메타문자 뒤에 ?를 붙이면 더 적은 갯수의 결과를 보여주는 것으로 변경됩니다.

?? : 0 또는 1개를 뜻하는데 0을 선호

+? : 1개 또는 이상을 뜻하는데 가능한 적은 갯수를 선호

*? : 0개 또는 이상을 뜻하는데 가능한 적은 갯수를 선호

{n,}? : n개 또는 이상을 뜻하는데 가능한 적은 갯수를 선호

{n,m}?: n개에서 m개 사이를 뜻하는데 가능한 적은 갯수를 선호

?를 활용한 조절의 사용예

아무 글자(.)가 모든 갯수가 가능한(*) 구성이

와

사이에 있는 경우입니다. .*과 .*?가 어떻게 다르게 동작하는지 확인해 보세요.

```
stri<-"<p> <em>안녕</em>하세요 </p><p>테스트입니다.</p>"
sub("<p>.*</p>","tar",stri)
```

```
## [1] "tar"
```

```
sub("<p>.*?</p>","tar",stri)
```

```
## [1] "tar<p>테스트입니다.</p>"
```

그룹

정규표현식에서는 글자 하나하나를 하나의 개체로 인식합니다. 예를 들어 abcabcabc같은 경우 지금 까지 배운 내용으로는 단순 매칭하는 방법밖에 없습니다. 이때 그룹을 사용하면 (abc)+이나 (abc){3}로 표현할 수 있습니다. ()는 괄호 안에 있는 글자 전체를 하나의 글자로 인식할 수 있게 해줍니다.

```
x <- c("abc","abcabc","abcabcadc","abcabcabc","adcabcabcabc")
grep("(abc){3}", x)
```

```
## [1] 4 5
```

그룹의 캡쳐 및 사용

그룹은 sub등 치환 기능을 사용할 때 더욱 빛을 발합니다. 찾는 패턴에서 그룹을 지어둔 내용은 순서대로 \\1, \\2의 방법으로 바꿀 패턴에서 사용할 수 있습니다.

```
x <- c("^ab", "ab", "abc", "ab 12")
gsub("(ab) 12", "\\1 34", x)
```

```
## [1] "^ab"   "ab"    "abc"   "ab 34"
```

또는의 사용

|는 or의 뜻으로 사용하는 글자입니다. 우선 단순 매칭에서 사용하는 경우입니다. ()과 함께 사용할 수도 있습니다.

```
x <- c("^ab", "ab", "ac", "abc", "abd", "abe", "ab 12")
grep("abc|abd", x)
```

```
## [1] 4 5
```

```
grep("a(c|bd)", x)
```

```
## [1] 3 5
```

함께 사용하는 함수

grep : 찾고자 하는 패턴이 있는 벡터의 위치를 결과로 줌
grepl : 찾고자 하는 패턴 인지를 TRUE, FALSE 벡터로 표현

sub : 찾고자 하는 첫번째 패턴을 두번째 인자로 바꿈
gsub : 찾고자 하는 모든 패턴을 두번째 인자로 바꿈

regexpr : 찾고자 하는 패턴의 글자내 시작점을 결과로 줌
gregexpr: 찾고자 하는 패턴의 글자내 위치를 모두 결과로 줌

dir : 찾고자 하는 패턴의 파일 이름을 결과로 줌

strsplit: 자르고자 하는 패턴으로 글자 데이터를 자름

apropos : Environment에 보여주지 않는 기본 객체들을 보여줌

- find : 객체가 어디에 포함되어있는지 보여줌

우편번호

우리나라는 새로운 방식인 "12345"와 "123-456"의 두 가지 방식으로 우편번호를 사용하고 있습니다.

```
^[0-9]{3}([0-9]{2}|-[0-9]{3})&
```

시각화 보러가기

주민등록번호

주민등록번호 또한 많이 사용하는 데이터입니다. 조건에 부합하는 데이터가 얼마나 되는지 확인하는데 유용합니다.

```
^([0-9]{2}(0[1-9]|1[0-2])(0[1-9]|[12][0-9]|3[01]))-[1-4][0-9]{6}$
```

시각화 보러가기

전화번호

다양한 입력방식의 전화번호를 전화번호로 인지하는 방법을 알아보겠습니다.

```
^\\(?[0-9]{2,3}\\)?[-. ]?[0-9]{3,4}[-. ]?[0-9]{4}$
```

시각화 보러가기
그룹과 gsub로 양식을 통일시킬 수도 있습니다.

```
gsub("^\\(?([0-9]{2,3})\\)?[-. ]?([0-9]{3,4})[-. ]?([0-9]{4})$",
      "(\\1) \\2-\\3",data)
```

이메일 주소

이메일은 중간에 @표시가 있으며 뒤에 서비스명 주소는 .com등 .을 포함하고 짧은 문자열로 구성되어 있습니다. co.kr같은 경우도 고려해야 합니다.

```
/^([a-z0-9_\\.-]+)@([0-9a-z\\.-]+)\.([a-z\\.]{2,6})$/
```

시각화 보러가기

인터넷 주소

인터넷 주소 또한 다양한 방법이 있습니다.

```
/^(https?:\\/\\/)?([\\da-z\\.-]+)\\.([a-z\\.]{2,6})([\\/[[:word:]]_\\.-]*)*\\/?$/
```

시각화 보러가기


stringr

소개

stringr은 tidyverse 패키지에 포함되어 있는 글자 조작용 패키지로 ICU라고 불리는 C library의 wrapper인 stringi 패키지를 기반으로 하고 있습니다.

총 패턴 매칭, 공백 관리, 로케일에 따라 동작이 다른 함수, 헬퍼의 4가지 카테고리의 함수를 제공합니다.

패턴 매칭

str_count : 찾고자 하는 패턴이 몇 개가 있는지 셈
str_detect : 찾고자 하는 패턴이 있는지 TRUE/FALSE로 반환
str_extract str_extract_all : 찾는 패턴을 뽑아서 출력
str_locate str_locate_all : 찾는 패턴의 글자내 시작점과 끝점을 출력
str_match str_match_all : 찾는 패턴을 뽑아주는데 캡쳐 그룹도 같이 제공
str_replace str_replace_all : 찾는 패턴을 다른 내용으로 바꿈
str_split str_split_fixed : 지정한 패턴으로 글자를 나눔
str_subset : 찾는 패턴이 있는 위치의 데이터를 출력
str_which : 찾는 패턴이 있는 데이터의 위치를 출력
str_view str_view_all : html로 매칭된 내용을 출력
fixed : 글자 그대로를 매칭할 것인지를 결정

공백 관리

str_pad : 글자의 좌, 우 혹은 양쪽에 띄어쓰기를 추가
str_trim : 글자의 좌, 우 혹은 양쪽에 있는 공백문자를 제거

로케일에 따라 동작이 다른 함수

str_order str_sort : 순서를 정렬
str_to_upper str_to_lower str_to_title : 알파벳의 경우 변경

헬퍼

str_c : 글자 벡터를 하나의 글자 데이터로 합침
str_conv : 글자의 인코딩을 조절
str_dup : 글자들을 반복하여 합침
str_length : 글자의 길이를 셈
str_replace_na : 자료형 NA를 글자 NA로 변경
str_trunc : 일정 글자 길이 이후를 줄임표로 자름
str_sub : 글자의 시작점과 끝점을 지정하여 추출

설치

stringr 패키지는 tidyverse 패키지에 포함되어 있는 패키지입니다.

```
if (!require(tidyverse)) install.packages("tidyverse")
```

```
## Loading required package: tidyverse

## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr

## Conflicts with tidy packages ----------------------------------------------

## filter(): dplyr, stats
## lag():    dplyr, stats
```

```
library(stringr)
```

글자의 길이

nchar()와 같이 길이를 결과로 주는 함수를 제공합니다. 한글도 지원하고, vector, factor 입력도 다 지원합니다.

```
str_length(c("abc","한글","ㅎㄱㅏ"))
```

```
## [1] 3 2 3
```

글자의 일부만 가져오기

위치(시작점와 끝점)를 기준으로 일부만 가져오는(subset) 기능을 제공합니다. 왼쪽의 첫번째 글자가 1의 위치이고 이후로 글자당 위치가 지정되며 오른쪽의 글자를 기준으로 세고 싶으면 음수로 지정하면 됩니다.

```
x <- c("abcdef", "ghifjk")
str_sub(x, 3, 3)
```

```
## [1] "c" "i"
```

```
str_sub(x, 3, 4)
```

```
## [1] "cd" "if"
```

```
str_sub(x, -4, -2)
```

```
## [1] "cde" "ifj"
```

일부분 바꾸기

일반적인 data.frame의 subset 동작처럼 일부분을 지정하고, 값을 선언함으로써 바꾸는 것이 동작합니다.

```
str_sub(x, 3, 3) <- "X"
x
```

```
## [1] "abXdef" "ghXfjk"
```

글자 반복해서 생성

```
x <- c("abcdef", "ghifjk")
str_dup(x, c(2, 3))
```

```
## [1] "abcdefabcdef"       "ghifjkghifjkghifjk"
```

공백문자 제거

글자를 다루는데 가장 많이 쓰는 기능으로 좌우의 공백문자(띄어쓰기, 엔터 등)를 제거해줍니다. 좌 또는 우를 선택할 수 있습니다.

```
x <- c("  a   ", "b   ",  "   c", "  c \n","\t   c  \n")
str_trim(x)
```

```
## [1] "a" "b" "c" "c" "c"
```

```
str_trim(x, "left")
```

```
## [1] "a   "  "b   "  "c"     "c \n"  "c  \n"
```

패딩

character vector의 length를 맞추기 위해서 띄어쓰기를 추가합니다.

```
x <- c("abc", "defghi")
str_pad(x, 10)
```

```
## [1] "       abc" "    defghi"
```

```
str_pad(x, 10, "both")
```

```
## [1] "   abc    " "  defghi  "
```

```
str_pad(x, 4)
```

```
## [1] " abc"   "defghi"
```

말줄임표현 만들기

truncated 란 잘라낸이라는 뜻이 있습니다. 텍스트가 길면 몇자 이상은 잘라내고 "..."으로 말줄임 표현을 사용할 수 있습니다. 텍스트에서는 이미지의 thumbnail과 비슷한 역할을 합니다.

```
x <- c("Short", "This is a long string")
x %>%
  str_trunc(10)
```

```
## [1] "Short"      "This is..."
```

알파벳 조절

대소문자를 3가지 함수로 조절할 수 있습니다.

```
x <- "I like horses."
str_to_upper(x)
```

```
## [1] "I LIKE HORSES."
```

```
str_to_title(x)
```

```
## [1] "I Like Horses."
```

```
str_to_lower(x)
```

```
## [1] "i like horses."
```

글자 순서

order와 sort가 글자를 기준으로 동작합니다.

```
x <- c("y", "i", "k")
str_order(x)
```

```
## [1] 2 3 1
```

```
str_sort(x)
```

```
## [1] "i" "k" "y"
```

패턴 매칭

정규표현식과 함께 쓰이는 패턴 매칭 함수들입니다.

```
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_subset(strings, phone)
```

```
## [1] "219 733 8965"
## [2] "329-293-8753"
## [3] "Work: 579-499-7527; Home: 543.355.3679"
```

```
str_detect(strings, phone)
```

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

```
str_count(strings, phone)
```

```
## [1] 0 1 1 2
```

위치 정보

패턴에 해당하는 글자의 위치를 시작과 끝으라는 형태로 결과를 줍니다. `str_locate_all(strings, phone)`를 실행해서 아래와 어떻게 결과가 다른지 확인해 보세요.

```
str_locate(strings, phone)
```

```
##       start end
## [1,]     NA  NA
## [2,]      1  12
## [3,]      1  12
## [4,]      7  18
```

패턴 추출

정규표현식을 활용하여 해당되는 패턴의 텍스트만 가져오는 함수 입니다. simplify 옵션이 어떻게 동작한 것인지 `str_extract_all(strings, phone)`를 실행하여 비교해 보세요.

```
str_extract(strings, phone)
```

```
## [1] NA              "219 733 8965" "329-293-8753" "579-499-7527"
```

```
str_extract_all(strings, phone, simplify = TRUE)
```

```
##      [,1]           [,2]
## [1,] ""             ""
## [2,] "219 733 8965" ""
## [3,] "329-293-8753" ""
## [4,] "579-499-7527" "543.355.3679"
```

그룹정보 보기

str_extract()는 해당되는 텍스트를 가져오는 함수라면 str_match()를 해당되는 것의 정보를 보여주는 함수입니다. 그래서 그룹으로 묶인 텍스트의 정보를 함께 파악할 수 있습니다.

```
str_match(strings, phone)
```

```
##      [,1]            [,2]  [,3]  [,4]
## [1,] NA              NA    NA    NA
## [2,] "219 733 8965" "219" "733" "8965"
## [3,] "329-293-8753" "329" "293" "8753"
## [4,] "579-499-7527" "579" "499" "7527"
```

```
str_match_all(strings, phone)
```

```
## [[1]]
##      [,1] [,2] [,3] [,4]
##
## [[2]]
##      [,1]            [,2]  [,3]  [,4]
## [1,] "219 733 8965" "219" "733" "8965"
##
## [[3]]
##      [,1]            [,2]  [,3]  [,4]
## [1,] "329-293-8753" "329" "293" "8753"
##
## [[4]]
##      [,1]            [,2]  [,3]  [,4]
## [1,] "579-499-7527" "579" "499" "7527"
```

글자 바꾸기

sub()와 gsub()에 해당하는 함수로 데이터가 맨 앞에 쓰인다는 점때문에 파이프연산자와 상성이 좋습니다.

```
str_replace(strings, phone, "XXX-XXX-XXXX")
```

```
## [1] "apple"
## [2] "XXX-XXX-XXXX"
## [3] "XXX-XXX-XXXX"
## [4] "Work: XXX-XXX-XXXX; Home: 543.355.3679"
```

```
str_replace_all(strings, phone, "XXX-XXX-XXXX")
```

```
## [1] "apple"
## [2] "XXX-XXX-XXXX"
## [3] "XXX-XXX-XXXX"
## [4] "Work: XXX-XXX-XXXX; Home: XXX-XXX-XXXX"
```

글자 나누기

하나의 데이터 상태로 있는 character를 기준 글자로 나누어 여러 개의 데이터로 바꾸는 동작입니다.

```
str_split("a-b-c", "-")
```

```
## [[1]]
## [1] "a" "b" "c"
```

```
str_split_fixed("a-b-c", "-", n = 2)
```

```
##      [,1] [,2]
## [1,] "a"  "b-c"
```

과제

1. recomen의 item.csv 파일을 불러와주세요.
    - cate_3_name 이 "립"으로 시작하는 제품을 판매하는 파트너사
    - cate_3_name 이 "소스"로 끝나는 제품
    - "/" 로 합쳐져 있는 cate_3_name를 모두 잘라서 유일한 제품명만 세면 모두 몇개 인가요?

2. `./data` 폴더의 `elevatorkr16.csv` 는 2016년 대한민국에 등록된 승강기 목록입니다. 이 데이터를 바탕으로 아래 문제를 풀어주세요.
    - (주)가 포함되고 빌딩으로 끝나는 건물명을 가진 건물의 수
    - 지역 이름 중 2글자+시로 이루어진 지역이 속한 도의 리스트
    - 승강기 종류중 마지막 글자가 용이 아닌 것의 리스트

과제

1. 아래 조건에 해당하는 정규표현식 패턴을 작성해주세요.
    - 국내 휴대전화번호
    - html의 주석 `<!-- 주석내용 -->`
    - IPv4
2. dabrp_class3 프로젝트를 연 후 `dir` 함수를 이용해서 아래 질문에 답해주세요.
    - `FolderForClass2` 폴더의 `db` 확장자 파일의 갯수
    - working diractory 에서 `R` 확장자 파일의 리스트
    - 모든 곳에서 `R` 확장자 파일의 리스트 (recursive 인자 참고)