Chapter 1

문자열과 날짜 및 시간

이번 챕터에서는 문자열 데이터와 시간과 관련된 데이터의 조작에 대해서 조금 더 알아보기로 할 것입니다. 문자열 데이터의 가장 대표적인 예로는 이름 또는 주소와 같이 정형화 된 형식으로 입력되지 않는 경우에 해당하며, 시간과 관련된 데이터는 생년월일, 입원일 또는 퇴원일과 같이 일반적인 문자열 데이터보다는 다소 정형화 된 형식을 가지고 있습니다. R에서는 이러한 문자열과 날짜를 처리하는데 있어 유용한 기능들을 제공하고 있습니다.

이전 챕터에서 다루었던 dart8.xls 데이터를 계속 활용하도록 합니다. 본 데이터는 여기 다운로드 링크를 눌러 다운받을 수 있습니다.

1.1 문자열 다루기

이전 챕터에서 다루었던 '다트8.5'이라는 데이터를 가지고 문자열 조작에 대한 예를 들어보도록 하겠습니다. 이를 위해서 '다트8.5' 데이터를 '다트8.7'라는 객체로 아래와 같이 저장합니다.

```
> 다트8.7 <- 다트8.5
> 다트8.7
        매출액        여행사 년도
30  57624282255     하나투어 2008
40  66245202818     하나투어 2010
77  19926326125   레드캡투어 2008
87  28589391364   레드캡투어 2010
104 26530405258     모두투어 2008
114 36466418562     모두투어 2010
149 16463319852         세중 2008
159 20282845926         세중 2010
168  5790815204   참좋은레저 2008
178 14717211161   참좋은레저 2010
191 12954724212 툿데관광개발 2008
201 12561340891 툿데관광개발 2010
232  4803096281     자유투어 2008
```

```
242 10455493093      자유투어 2010
275  1012965573    비티앤아이 2008
285  4986588998    비티앤아이 2010
>
```

이 데이터에서는 매출액은 수치형 변수임을 알고 있고, 여행사는 문자열 변수이고, 년도는 수치형이면서도 요인형 변수입니다. 그리고, first 와 last 라는 변수은 논리형 변수입니다. 만약, 아래와 같이 년도라는 데이터가 없다면 분석자는 여행사 데이터가 중복이 된 것인지 아닌지를 판단하기가 불확실하기 때문에 데이터처리에 곤란함을 느낄 것입니다.

```
> 다트8.7 <- 다트8.7[c(TRUE, TRUE, FALSE)]
> 다트8.7
          매출액        여행사
30  57624282255      하나투어
40  66245202818      하나투어
77  19926326125    레드캡투어
87  28589391364    레드캡투어
104 26530405258      모두투어
114 36466418562      모두투어
149 16463319852          세중
159 20282845926          세중
168  5790815204    참좋은여저
178 14717211161    참좋은여저
191 12954724212 롯데관광개발
201 12561340891 롯데관광개발
232  4803096281      자유투어
242 10455493093      자유투어
275  1012965573    비티앤아이
285  4986588998    비티앤아이
>
```

따라서, 매출액의 값이 다를지라도 여행사 변수에 대해서 분석자는 추후에 데이터에 대한 중복확인, 다른 자료의 유입, 혹은 다른 시점으로부터의 입력인가에 대한 확인하고자 임시적으로 현재 '여행사'변수의 값이 중복되었다면 .1 이라는 값을 붙여 두고 싶을 수 있습니다. (독자는 추후에 R에서 중복된 값들은 보통 동일한 값의 뒤에 .1 이 붙는다는 것을 보게 될 것입니다. 그리고, 이미 이전 챕터에서 행이름이 .1, .2, .3, ... 과 같이 생성된다는 것을 보았습니다. 아래에서 '여행사2'라는 변수를 살펴보시길 바랍니다.

```
> 다트8.7$여행사2 <- ifelse(duplicated(다트8.7$여행사), paste(다트8.7$여행사, 1, sep="."),paste(다트8.7$
> 다트8.7
          매출액        여행사        여행사2
30  57624282255      하나투어       하나투어
40  66245202818      하나투어     하나투어.1
77  19926326125    레드캡투어     레드캡투어
```

```
87  28589391364    레드 캡투 어    레드 캡투 어.1
104 26530405258      모 두 투 어       모 두 투 어
114 36466418562      모 두 투 어     모 두 투 어.1
149 16463319852          세중           세중
159 20282845926          세중         세중.1
168  5790815204    참 좋 은 레 져     참 좋 은 레 져
178 14717211161    참 좋 은 레 져   참 좋 은 레 져.1
191 12954724212 툿 데 관 광 개 발   툿 데 관 광 개 발
201 12561340891 툿 데 관 광 개 발 툿 데 관 광 개 발.1
232  4803096281      자 유 투 어       자 유 투 어
242 10455493093      자 유 투 어     자 유 투 어.1
275  1012965573    비 티 앤 아 이     비 티 앤 아 이
285  4986588998    비 티 앤 아 이   비 티 앤 아 이.1
>
```

여기에서 사용된 함수 duplicated()는 중복을 확인하는데 사용되었습니다 (이전 챕터에서 설명했습니다). 함수 paste()의 기능은 주어진 인자들을 문자열로 묶는 것이며, 이 때 주어진 모든 인자들은 문자열로 강제변환이 됨을 알아두면 좋습니다. 즉, 아래와 같이 숫자 1을 입력할지라도 paste()함수 내부에서 이를 문자열로 처리하여 결과를 돌려주게 됩니다.

```
> paste(다트8.7$여행사, 1, sep=".")
 [1] "하 나 투 어.1"     "하 나 투 어.1"     "레 드 캡 투 어.1"   "레 드 캡 투 어.1"
 [5] "모 두 투 어.1"     "모 두 투 어.1"     "세중.1"             "세중.1"
 [9] "참 좋 은 레 져.1"  "참 좋 은 레 져.1"  "툿 데 관 광 개 발.1" "툿 데 관 광 개 발.1"
[13] "자 유 투 어.1"     "자 유 투 어.1"     "비 티 앤 아 이.1"   "비 티 앤 아 이.1"
```

함수 paste()에서 사용되는 지정된 인자 sep은 사용된 인자들이 어떤 구분기호를 이용하여 결합할 것인지를 알려줍니다. 위에서와 같이 어떤 독자는 . 를 구분기호로서 사용하고 싶을 수도 있고, 어떤 독자는 - 를 선호할 수 있습니다.

```
> paste(다트8.7$여행사, 1, sep="-")
 [1] "하 나 투 어-1"     "하 나 투 어-1"     "레 드 캡 투 어-1"   "레 드 캡 투 어-1"
 [5] "모 두 투 어-1"     "모 두 투 어-1"     "세중-1"             "세중-1"
 [9] "참 좋 은 레 져-1"  "참 좋 은 레 져-1"  "툿 데 관 광 개 발-1" "툿 데 관 광 개 발-1"
[13] "자 유 투 어-1"     "자 유 투 어-1"     "비 티 앤 아 이-1"   "비 티 앤 아 이-1"
```

어떤 독자는 아무런 공백없이 연결하고자 할 수도 있습니다.

```
> paste(다트8.7$여행사, 1, sep="")
 [1] "하 나 투 어1"     "하 나 투 어1"     "레 드 캡 투 어1"   "레 드 캡 투 어1"
 [5] "모 두 투 어1"     "모 두 투 어1"     "세중1"             "세중1"
 [9] "참 좋 은 레 져1"  "참 좋 은 레 져1"  "툿 데 관 광 개 발1" "툿 데 관 광 개 발1"
[13] "자 유 투 어1"     "자 유 투 어1"     "비 티 앤 아 이1"   "비 티 앤 아 이1"
>
```

이렇게 문자열을 결합하는 기능을 가진 함수 paste()를 이용하여 아래와 같이 변수명을 생성할때 유용하게 쓰일 수도 있습니다.

```
> paste("V", 1:3, sep="")
[1] "V1" "V2" "V3"
>
```

또 다른 유용한 경우는 아래와 같이 문자와 숫자의 조합을 형성하기 쉽습니다. (여기에서 사용된 함수 outer()는 수학/확률 이라는 챕터에서 구구단을 만드는 예제에서 설명되어 있습니다).

```
> outer(LETTERS[1:3], 1:3, paste, sep="-")
     [,1]  [,2]  [,3]
[1,] "A-1" "A-2" "A-3"
[2,] "B-1" "B-2" "B-3"
[3,] "C-1" "C-2" "C-3"
```

이제 좀 더 일반적으로 문자열을 다루는 방법에 대해서 알아보도록 하겠습니다. 먼저, 아래와 같이 현재 R 세션에 로드되어 있는 기본 패키지들을 확인하도록 합니다.

```
> str.ex <- getOption("defaultPackages")
> str.ex
[1] "datasets"  "utils"     "grDevices" "graphics"  "stats"     "methods"
>
```

만약, str.ex 이라는 문자열벡터를 이루는 각 개별 구성요소들에 대한 문자열의 길이를 알고 싶다면 아래와 같이 함수 nchar()를 사용하세요.

```
> nchar(str.ex)
[1] 8 5 9 8 5 7
>
```

갑자기 어떤 독자가 현재 문자열은 영어로 되어 있어서 잘 되는것이 아닌가하고 질문을 합니다. 그래서, 이전의 '다트8.5' 데이터에서 '여행사'변수의 값을 이용하여 한글도 잘 되는가를 확인해 봅니다.

```
> 다트8.7 <- 다트8.5
> str.ex1 <- unique(다트8.7$여행사)
> str.ex1
[1] 하나투어      레드캡투어    모두투어      세중          참좋은데져
[6] 롯데관광개발  자유투어      비티앤아이
8 Levels: 하나투어 레드캡투어 모두투어 세중 참좋은데져 ... 비티앤아이
>
> str.ex1 <- as.character(unique(다트8.7$여행사))
> str.ex1
[1] "하나투어"      "레드캡투어"    "모두투어"      "세중"          "참좋은데져"
[6] "롯데관광개발" "자유투어"      "비티앤아이"
```

```
> nchar(str.ex1)
[1] 4 5 4 2 5 6 4 5
>
```

한글에서도 문자열의 길이를 알려주는 함수가 잘 작동함을 알게 되었으므로, 이제는 문자열의 어떤 위치로부터 어떤 위치까지 뽑아 낼 수 있는 기능에 대해서 알아봅니다.

먼저 위에서 사용했던 문자열 str.ex 를 불러오고, 이 문자열을 구성하는 각 요소들의 첫번째 위치부터 세번째까지의 문자들만 골라 내고 싶다면 아래와 같이 함수 substr()을 이용합니다.

```
> str.ex
[1] "datasets"  "utils"     "grDevices" "graphics"  "stats"     "methods"
> substr(str.ex, 1, 3)
[1] "dat" "uti" "grD" "gra" "sta" "met"
```

이제 함수 nchar()를 함께 사용하여 각 구성요소의 첫번째 문자부터 맨 마지막 문자까지 뽑아냅니다. 즉, 원래의 문자열과 동일한 값을 얻어야 할 것입니다.

```
> substr(str.ex, 1, nchar(str.ex))
[1] "datasets"  "utils"     "grDevices" "graphics"  "stats"     "methods"
```

어떤 독자는 문자열의 일부분을 추출하기 보다는 어떤 특정 구분자에 의해서 문자열 자체를 쪼개기를 희망할 수있습니다. 이를 위해서 아래와 같이 현재 세션에 놓인 모든 탐색경로를 이용합니다.

```
> str.ex2 <- search()
> str.ex2
 [1] ".GlobalEnv"        "package:XLConnect" "package:rJava"
 [4] "package:stats"     "package:graphics"  "package:grDevices"
 [7] "package:utils"     "package:datasets"  "package:methods"
[10] "Autoloads"         "package:base"
```

이 탐색경로를 잘 보면 str.ex2의 대다수의 구성요소들이 package 라는 이름 뒤에 콜론 (:)이 놓여지고 그 뒤에 패키지명이 놓인다는 것을 알 수 있습니다. 여기에서 하고자 하는 것은 str.ex2을 이루는 각 구성요소를 ":"라는 구분자를 이용하여 분리하는 것입니다. 이 경우에는 아래와 같은 함수 strsplit()을 사용할 수 있습니다.

```
> strsplit(str.ex2, ":")
[[1]]
[1] ".GlobalEnv"

[[2]]
[1] "package"   "XLConnect"

[[3]]
[1] "package" "rJava"
```

```
[[4]]
[1] "package" "stats"

[[5]]
[1] "package"  "graphics"

[[6]]
[1] "package"   "grDevices"

[[7]]
[1] "package" "utils"

[[8]]
[1] "package"  "datasets"

[[9]]
[1] "package" "methods"

[[10]]
[1] "Autoloads"

[[11]]
[1] "package" "base"
```

그런데, strsplit()함수의 결과물은 리스트 형태임을 알 수 있으므로, 보기 편하게 아래와 같이 do.call()함수를 이용해봅니다.

```
> do.call(rbind, strsplit(search(), ":", fixed=FALSE))
      [,1]         [,2]
 [1,] ".GlobalEnv" ".GlobalEnv"
 [2,] "package"    "XLConnect"
 [3,] "package"    "rJava"
 [4,] "package"    "stats"
 [5,] "package"    "graphics"
 [6,] "package"    "grDevices"
 [7,] "package"    "utils"
 [8,] "package"    "datasets"
 [9,] "package"    "methods"
[10,] "Autoloads"  "Autoloads"
[11,] "package"    "base"
>
```

여기까지 우리는 문자열의 부분추출, 결합, 그리고 분리에 대해서 보았습니다. 어떤 독자는 치환은 어떻게 할 수 있는지에 대해서 궁금해 할 것입니다. 위에서 사용한 문자열 str.ex2 에서 package 라는 문자를

한국어로 "패키지"로 바꾸고 싶다면 아래와 같이 함수 gsub() 를 이용할 수 있습니다.

```
> str.ex2
 [1] ".GlobalEnv"        "package:XLConnect" "package:rJava"
 [4] "package:stats"     "package:graphics"  "package:grDevices"
 [7] "package:utils"     "package:datasets"  "package:methods"
[10] "Autoloads"         "package:base"
> gsub("package", "패키지", str.ex2)
 [1] ".GlobalEnv"       "패키지:XLConnect" "패키지:rJava"     "패키지:stats"
 [5] "패키지:graphics"  "패키지:grDevices" "패키지:utils"     "패키지:datasets"
 [9] "패키지:methods"   "Autoloads"        "패키지:base"
>
```

그런데 이러한 치환과정은 먼저 주어진 문자열을 탐색하는 것으로부터 이루어집니다. 따라서, 독자가 R 을 이용하여 어떤 문자열을 탐색하고 싶다면 아래와 같은 grep()함수를 사용해 볼 수 있습니다. 먼저 letters 라는 예약어는 알파벳 a부터 z까지의 레터들을 미리 저장해두고 있습니다.

```
> letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

여기에서 a 라는 문자가 어디에 위치하는지, 또는 e 라는 문잔는 어디에 위치하는지알고 싶다면 함수 grep()의 사용은 유용합니다.

```
> grep("a", letters)
[1] 1
> grep("e", letters)
[1] 5
```

이렇게 얻어진 위치정보를 이용하여 아래와 같이 e 라는 문자를 찾을 수 있습니다.

```
> letters[grep("e", letters)]
[1] "e"
```

또한 grepl()이라는 함수가 있는데 이는 논리값으로 그 결과를 돌려줍니다. grep뒤에 l은 logical을 의미합니다.

```
> grepl("e", letters)
 [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[25] FALSE FALSE
```

함수 grep은 찾고자 하는 문자를 탐색하여 검색어를 포함하는 모든 문자열의 위치를 알려줍니다. 이를 이해하기 위해서 아래의 예제를 보길 바랍니다.

```
> txt <- c("R", "is", "free", "software", "and", "your", "friend")
> txt
[1] "R"        "is"       "free"     "software" "and"      "your"     "friend"
```

이러한 문자열 벡터에서 o 라는 글자를 포함하는 문자열은 아래와 같이 얻을 수 있습니다.

```
> grep("o", txt)
[1] 4 6
> grepl("o", txt)
[1] FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE
> txt[grep("o", txt)]
[1] "software" "your"
>
```

이제 한글은 어떻게 작동하는지 확인해보도록 합니다.

```
> txt.ko <- c("김지운", "강사", "남녀", "연애", "정말?", "ㅋㅋ", "그랬어?", "맞 아", "오빠", "아잉")
> txt.ko
 [1] "김지운"  "강사"    "남녀"    "연애"    "정말?"   "ㅋㅋ"    "그랬어?"
 [8] "맞아"    "오빠"    "아잉"
>
```

여기에서 “아”라는 글자가 들어간 문자열을 찾고 싶을때는 아래와 같이 할 수 있습니다.

```
> grep("아", txt.ko)
[1]  8 10
```

같은 방법으로 “김” 또는 “애” 라는 글자가 들어간 문자열도 찾을 수 있습니다.

```
> grep("[김애]", txt.ko)
[1] 1 4
```

그런데, 아래와 같이 특수문자를 사용할 때는 주의를 해야합니다.

```
> grep("^?", txt.ko)
 [1]  1  2  3  4  5  6  7  8  9 10
> grep("^", txt.ko)
 [1]  1  2  3  4  5  6  7  8  9 10
> grep("*", txt.ko)
 [1]  1  2  3  4  5  6  7  8  9 10
>
```

주어진 특수문자를 올바르게 찾기 위해서는 꼭 대괄호 안에 위치시켜주어야 합니다.

```
> grep("[?]", txt.ko)
[1] 5 7
```

그런데, 갑자기 독자가 지금 사용한 문자열을 하나의 문자열로 만들고 싶어합니다. 이런 경우 역시 paste()함수를 활용할 수 있습니다.

```
> txt
[1] "R"        "is"       "free"     "software" "and"      "your"     "friend"
> paste(txt, collapse=" ")
[1] "R is free software and your friend"
```

이전에 사용했던 sep 인자는 문자열 벡터의 구성요소 내에서 작용하는 것이고, collapse는 구성요소간에 작용하는 것임을 알고 계시면 도움이 됩니다.

```
> paste(txt, collapse="-")
[1] "R-is-free-software-and-your-friend"
```

한글 문자열 역시 잘 됨을 확인할 수 있습니다.

```
> paste(txt.ko, collapse="-")
[1] "김지윤-강사-남너-연애-정말?-ㅋㅋ-그랬어?-맞아-오빠-아잉"
> paste(txt.ko, collapse=" ")
[1] "김지윤 강사 남너 연애 정말? ㅋㅋ 그랬어? 맞아 오빠 아잉"
>
```

문자열 섹션을 마치기 전에 아래와 같은 사항을 알아두시면 도움이 될 것 같아 남겨둡니다.

문자열을 출력할 때는 주로 print()와 cat()을 이용합니다. 이 두함수의 차이는 이스케이프 시퀀스가 적용이 되는가 안되는가의 차이입니다. 이스케이프 시퀀스라는 것은 문자열을 사람이 잘 읽을 수있도록 도와주는 일종의 특수기호이며, 이는 컴퓨터만이 이해하도록 하고 출력시에는 보여주지 않도록 처리됩니다. cat()이라는 문자는 이러한 이스케이스 시퀀스를 아래와 같이 처리해서 보여주지만, print()함수는 그러한 과정없이 그대로 보여줍니다.

```
> cat("Hi \nHello \n")
Hi
Hello
> print("Hi \nHello \n")
[1] "Hi \nHello \n"
```

만약, 문자의 대소문자를 변경하고 싶다면 tolower()와 toupper()함수를 이용해보세요.

```
> txt1 <- paste(txt, collapse=" ")
> txt1
[1] "R is free software and your friend"
> txt1.1 <- toupper(txt1)
> txt1.1
[1] "R IS FREE SOFTWARE AND YOUR FRIEND"
> txt1.2 <- tolower(txt1.1)
> txt1.2
[1] "r is free software and your friend"
>
```

R에서는 Perl에서 제공하는 regular expression을 사용합니다. (이 부분은 따로 제공해주는게 좋을 것 같음).

1.2 날짜와 시간 다루기

데이터 실무라는 챕터에서 다루었던 dart8.xls 이라는 데이터셋에서 여행사별 매출액 변수는 년도와 분기별로 기록이 되어 있음을 알 수 있었습니다. R에서는 이러한 시간과 날짜를 다루는 기능들을 제공하고 있습니다. dart8.xls데이터는 R의 이러한 기능들을 설명하는데 다소 부족함이 있으므로, 일반적인 설명하기 위해 임의적으로 데이터를 필요에 따라 생성하여 설명하도록 하겠습니다.

시간과 날짜를 다룰 때 제일 먼저 알아야 할 것은 로케일 설정입니다. 로케일 (locale)이라는 것을 쉽게 이해하기 위해서는 지역설정이라고 생각하시면 됩니다. 즉, 사용자의 국가별 언어, 화폐단위, 그리고 시간대를 설정할 수 있는 변수입니다.

따라서, R을 한국어로 사용하시는 분들은 이러한 로케일 정보에 따라서 한글, 원(화폐단위), 년-월-일 (시간정보)를 사용하게 됩니다. (이곳에서는 로케일과 관련하여 날짜와 시간에 대해서만 이야기 하므로, 한글에 대해서는 “환경설정과 유틸리티”라는 챕터에서 “한글과 UTF-8 인코딩”이라는 섹션을 살펴보시길 바랍니다).

이러한 로케일에 대한 정보는 아래와 같이 Sys.getlocale()이라는 함수를 통하여 얻을 수 있습니다. 그런데, 이러한 로케일의 설정은 R과 관계없이 운영체제로부터 그 정보를 얻어온다는 것을 아셔야 합니다.

```
> Sys.getlocale()
[1] "LC_CTYPE=en_CA.UTF-8;LC_NUMERIC=C;LC_TIME=en_CA.UTF-8;LC_COLLATE=en_CA.UTF-8;LC_MONETARY=en_CA.UTF-
```

이렇게 보니까 눈에 잘 안들어와서, 위의 “문자열 다루기”섹션에서 배웠던 strsplit()함수를 이용한뒤 unlist()함수를 적용해봅니다.

```
> unlist(strsplit(Sys.getlocale(), ";"))
 [1] "LC_CTYPE=en_CA.UTF-8"       "LC_NUMERIC=C"
 [3] "LC_TIME=en_CA.UTF-8"        "LC_COLLATE=en_CA.UTF-8"
 [5] "LC_MONETARY=en_CA.UTF-8"    "LC_MESSAGES=en_CA.UTF-8"
 [7] "LC_PAPER=C"                 "LC_NAME=C"
 [9] "LC_ADDRESS=C"               "LC_TELEPHONE=C"
[11] "LC_MEASUREMENT=en_CA.UTF-8" "LC_IDENTIFICATION=C"
>
```

눈에 보기 좋아 이해하기가 쉽군요. (이는 단순히 복습차원에서 한 번 내용을 적어본것입니다).

현재 저의 운영체제에 있는 로케일 설정설정은 en_CA 로 캐나다에서 사용되는 언어를 사용하고 있습니다. 그럼, 이 정보가 과연 운영체제로부터 전달되었는지 R의 콘솔이 아닌 쉘상에서 확인을 해 봅니다.

```
gnustats@CHL072:~$ locale
LANG=en_CA.UTF-8
LANGUAGE=en_CA:en
LC_CTYPE="en_CA.UTF-8"
LC_NUMERIC="en_CA.UTF-8"
LC_TIME="en_CA.UTF-8"
LC_COLLATE="en_CA.UTF-8"
LC_MONETARY="en_CA.UTF-8"
LC_MESSAGES="en_CA.UTF-8"
```

```
LC_PAPER="en_CA.UTF-8"
LC_NAME="en_CA.UTF-8"
LC_ADDRESS="en_CA.UTF-8"
LC_TELEPHONE="en_CA.UTF-8"
LC_MEASUREMENT="en_CA.UTF-8"
LC_IDENTIFICATION="en_CA.UTF-8"
LC_ALL=
gnustats@CHL072:~$
```

이러한 저의 환경에서 만약 한국어 로케일이 없다면 한국어로 볼 수 없고, 한국어 날짜를 표기 할 수 없을 것입니다. 따라서, 한국어 로케일이 있는지 확인해 봅니다.

```
gnustats@CHL072:~$ locale -a
C
C.UTF-8
en_AG
en_AG.utf8
en_AU.utf8
en_BW.utf8
en_CA.utf8
en_DK.utf8
en_GB.utf8
en_HK.utf8
en_IE.utf8
en_IN
en_IN.utf8
en_NG
en_NG.utf8
en_NZ.utf8
en_PH.utf8
en_SG.utf8
en_US.utf8
en_ZA.utf8
en_ZM
en_ZM.utf8
en_ZW.utf8
es_CO.utf8
ko_KR.utf8
POSIX
zh_CN.utf8
zh_SG.utf8
gnustats@CHL072:~$
```

내용을 쭈루룩 읽다보니 아하! ko_KR.utf8 이라는 것이 보입니다. 이렇게 ko_KR.utf8이 존재한다면 한국어 관련 로케일을 표시하는데 문제가 없습니다. 만약, 없는 분이 있다면 아래의 명령어를 이용하여 꼭 추가를 하신뒤에 사용을 하시길 바랍니다.

```
gnustats@CHL072:~$ sudo localedef -f UTF-8 -i ko_KR ko_KR.UTF-8
```

이 명령어는 사용가능한 로케일들을 보관하고 있는 /usr/share/i18n/locales/locale-archive 라는 파일에 ko_KR.utf8이 추가됩니다. 따라서, locale -a를 실행하게 되면 한국어 로케일을 볼 수 있게 됩니다. 명령어에 사용된 -f 라는 인자의 의미는 /usr/share/i18n/charmaps/UTF-8.gz 라는 캐릭터 맵 (즉, 문자의 기호와 인코딩을 매칭하는 정의)를 이용하며, -i 라는 인자는 ko_KR 이라는 UTF-8로 로케일을 컴파일 할 수 있도록 지정하는 것입니다. 본래 한국어 언어 처리 부분은 /usr/share/lcoale/ko/LC_MESSAGES 에 프로그램에 사용되는 메시지를 담고 있으며, 해당 프로그램 설치시 make install 을 통해서 관련위치에 복사되는 것입니다. 따라서, 리눅스에서 한국어 메시지 대신 영어로 사용을 하고자 하시는 분들은 LC_MESSAGES 카테고리를 C 또는 POSIX로 변경해주면 됩니다. 이러한 방법이 그대로 R에서 한국어를 출력하는 방식으로 되어 있습니다.

로케일을 설명하면서 잠시 삼천포로 빠졌으나 위의 내용에 대한 이해는 R을 사용하는데 반드시 도움을 주게 됩니다. 다시 날짜와 시간관리에 대한 본론으로 돌아옵니다. R은 날짜와 시간을 크게 두가지로 구분합니다. 하나는 2013년 05월 16일과 같이 시간정보 없이 날짜만을 다루는 Date 라는 클래스를 이용하는 것이고, 그리고 다른 하나는 2013년 05월 16일 오후 06시 28분 33초 와 같이 날짜와 시간을 동시에 다루는 POSIXlt와 POSIXct 라는 클래스를 이용하는 것입니다.

먼저 아래와 같이 날짜를 출력해주는 Sys.Date()을 이용해봅니다.

현재시간을 출력해주는 함수 Sys.Date()에 대해서 알아봅니다.

```
> cdt <- Sys.Date()
> cdt
[1] "2013-05-19"
```

여기에서 한가지 꼭 알아둘 것은 R이 날짜 데이터 처리에 사용하는 표준형식은 International Organization for Standardization 이라는 ISO 8601 표준방식에서 지정되어 있는 대시기호 (-)를 이용한 "YYYY-MM-DD"(년-월-일)이라는 것입니다. (추후에 ISOdate()라는 함수에 대해서 설명할 것입니다). 꼭 기억해주세요.

```
> str(cdt)
 Date[1:1], format: "2013-05-19"
> class(cdt)
[1] "Date"
>
```

이번에는 시간을 출력해주는 Sys.time()에 대해서 알아봅니다.

```
> ct <- Sys.time()
> ct
[1] "2013-05-19 12:08:22 CST"
> str(ct)
 POSIXct[1:1], format: "2013-05-19 12:08:22"
> class(ct)
```

```
[1] "POSIXct" "POSIXt"
>
```

그러나, 본래 날짜와 시간 출력은 "YYYY-MM-DD" (년도-월-일)이라는 표준형식을 가진 문자열을 Date와 POSIXct라는 클래스의 형태로 변환하여 출력하는 것입니다. 따라서, 문자열이 아래와 같은 문자열을 Date 형식으로 변환이 가능합니다. 아래를 살펴보세요.

```
> "2013-05-20-19"
[1] "2013-05-20-19"
> str <- "2013-05-19"
> is.character(str)
[1] TRUE
> class(str)
[1] "character"
```

문자형 객체임을 확인했습니다. 이제 날짜로 변경해봅니다.

```
> corced.str <- as.Date(str)
> corced.str
[1] "2013-05-19"
> class(corced.str)
[1] "Date"
>
```

그러나, 한국어 시간표기법에 대해서는 다소 주의가 필요하기 때문에 개인적으로는 날짜를 다룰때 한국어 날짜표기법인 "XXXX년 XX월 XX일"이라는 형식은 피하고 대시 (-)를 이용해주시길 바랍니다.

```
> as.Date("2013년05월19일")
Error in charToDate(x) :
  character string is not in a standard unambiguous format
>
```

앗! 에러가 납니다 (위의 메시지는 한국어로 "다음에 오류charToDate(x) : 문자열이 표준서식을 따르지 않습니다" 이라고 나올 것입니다). 그 이유는 위에서 설명한 로케일 때문입니다. 그런데, 아쉽게도 이 문제는 로케일을 ko_KR.UTF-8 로 변경해도 제대로 표시되지 않습니다.

```
> Sys.setlocale("LC_TIME", "ko_KR.UTF-8")
[1] "ko_KR.UTF-8"
> as.Date("2013년05월19일")
Error in charToDate(x) :
  character string is not in a standard unambiguous format
```

그럼 한국날짜를 년/월/일을 이용해서 표현할 수는 없나요? 아래와 같이 format 옵션을 함께 조정해주세요.

```
> as.Date("2013년05월19일", "%Y년%m월%d일")
[1] "2013-05-19"
>
```

그럼 표준형식으로 출력되는 것을 확인할 수 있을 것입니다. 조금더 살펴봅니다.

```
> as.Date("05월19일2013년", "%Y년%m월%d일")
[1] NA
```

여기에서 NA 로 나오는 이유는 "05월19일2013년"의 입력순서는 월-일-년의 순서인데 출력은 년-월-일로 하라고 했기 때문입니다. 따라서, format 인자의 형식을 올바르게 아래와 같이 합니다.

```
> as.Date("05월19일2013년", "%m월%d일%Y년")
[1] "2013-05-19"
>
```

이렇게 format 인자를 다루는 것은 한국 로케일외에도 일반적으로 적용되는 규칙입니다. 아래를 살펴보세요.

```
> as.Date("2013/05/19", format="%Y-%m-%d")
[1] NA
> as.Date("2013/05/19", format="%Y/%m/%d")
[1] "2013-05-19"
> as.Date("2013-05-19", format="%Y/%m/%d")
[1] NA
> as.Date("2013-05-19", format="%Y%m%d")
[1] NA
> as.Date("2013년05월19일", format="%Y/m/%d")
[1] NA
> as.Date("2013년05월19일", format="%Y/%m/%d")
[1] NA
> as.Date("2013년05월19일", format="%Y년%m월%d")
[1] "2013-05-17"
```

현재 한국어로 된 R을 사용할때 가장 많이 겪는 경우는 날짜 데이터를 불러올때 와장창 NA로 나오는 경우 일 것입니다. 이것은 원래의 데이터가 입력될때 영어권 로케일 상태에서 입력이 된경우가 대다수입니다. 아래를 살펴보시면 이해가 가실 것입니다.

먼저 아래와 같이 영어권 로케일임을 확인합니다.

```
> Sys.getlocale("LC_TIME")
[1] "en_CA.UTF-8"
>
```

이제 아래와 같이 날짜 데이터를 입력하고 날짜가 나오는지 확인합니다.

```
> x <- c("1MAY2013", "2MAY2013", "3MAY2013")
> x
[1] "1MAY2013" "2MAY2013" "3MAY2013"
> class(x)
[1] "character"
> xc <- as.Date(x, format="%d%B%Y")
> xc
[1] "2013-05-01" "2013-05-02" "2013-05-03"
> class(xc)
[1] "Date"
```

이제 로케일을 한국으로 변경합니다.

```
> Sys.setlocale("LC_TIME", "ko_KR.UTF-8")
[1] "ko_KR.UTF-8"
> xc2 <- as.Date(x, format="%d%B%Y")
> xc2
[1] NA NA NA
>
```

따라서, 이와 같은 경우는 Sys.setlocale()함수를 이용하여 LC_TIME 이라는 환경설정 변수를 영어권국가로 변경한뒤 데이터를 읽으면 잘 수행됩니다. 그런데, 어느 영어권 국가인지 잘 모르겠다면 그냥 C 라고 변경하세요.

여기까지는 시간 없이 날짜만 다루었습니다. 이제 시간을 초단위까지 다루는 POSIX에 대해서 알아봅니다. 그런데, 이 클래스는 POSIXlt와 POSIXct 라는 두개의 종류를 가지고 있습니다.

먼저 아래와 같이 POSIXct 클래스를 봅니다.

```
> ct2 <- as.Date(Sys.Date())
> ct2
[1] "2013-05-19"
> cdt <- Sys.time()
> class(cdt)
[1] "POSIXct" "POSIXt"
> attributes(cdt)
$class
[1] "POSIXct" "POSIXt"
>
```

위에서 보는 바와 같이 아무런 속성을 가지고 있지 않습니다. 무엇이 POSIXct 클래스가 POSIXlt 클래스와 무엇이 다른지 봅니다.

```
> cdt2 <- as.POSIXlt(cdt)
> cdt2
[1] "2013-05-19 12:58:58 CST"
```

```
> attributes(cdt2)
$names
[1] "sec"   "min"   "hour"  "mday"  "mon"   "year"  "wday"  "yday"  "isdst"

$class
[1] "POSIXlt" "POSIXt"

$tzone
[1] ""    "CST" "CST"
>
```

아하! 둘 다 모두 시간을 초단위까지 다루지만, POSIXlt는 1970년 1월 1일 00시 00분 00초를 기준으로 현재의 날짜와 시간을 초단위로 기억해 두는 반면 POSIXct는 'sec', 'min', 'hour', 'mday', 'mon', 'year', 'wday' 'yday', 'isdst'라는 아홉개의 구성요소를 가진 리스트로 되어 있습니다.

그럼, 저기 있는 POSIXlt 클래스의 속성들이 가진 것들 대체 무엇을 의미할까요? "sec", "min", "hour"은 영어그대로 초,분,시각이고, 'mday'는 한 달중에 몇번째 날짜이며, 'year'는 1990년 이래로 몇 번째 해이며, 'wday'는 1주일중 몇 번째 날짜인지 알려줍니다. 아래와 같이 확인해 봅니다.

```
> cdt2$sec
[1] 58.97329
> cdt2$yday
[1] 138
>
```

따라서, 이러한 정보를 이용하면 보다 수월하게 시간 객체들을 다룰 수 있습니다.

그런데, 실제적으로는 아마 제일 필요한 것이 언제부터 언제까지의 시간객체를 만들어 내는 것일 수 있습니다. 이럴때 seq() 함수를 또 이용이 가능한데 주의할 점이 있습니다. 아래를 살펴보세요.

```
> seq(from="2013-05-17", to="2013-05-29")
Error in from:to : NA/NaN argument
In addition: Warning messages:
1: In seq.default(from = "2013-05-17", to = "2013-05-29") :
  NAs introduced by coercion
2: In seq.default(from = "2013-05-17", to = "2013-05-29") :
  NAs introduced by coercion
>
```

이럴 때는 by 옵션의 단위를 잘 선택해야 합니다. 일반벡터는 1이라는 숫자를 다루지만, 여기에서 우리는 날짜를 다루기 때문에 기본단위는 년/월/일 일것입니다.

일단위로 생성해 봅니다.

```
> seq(from=as.Date("2013-05-17"), to=as.Date("2013-05-29"), by=1)
 [1] "2013-05-17" "2013-05-18" "2013-05-19" "2013-05-20" "2013-05-21"
 [6] "2013-05-22" "2013-05-23" "2013-05-24" "2013-05-25" "2013-05-26"
```

```
[11] "2013-05-27" "2013-05-28" "2013-05-29"
>
```

년단위로 생성해봅니다.

```
> seq(from=as.Date("2013-05-17"), to=as.Date("2016-05-29"), by="year")
[1] "2013-05-17" "2014-05-17" "2015-05-17" "2016-05-17"
```

월단위도 한 번 해봅니다.

```
> seq(from=as.Date("2013-05-17"), to=as.Date("2016-05-29"), by="month")
 [1] "2013-05-17" "2013-06-17" "2013-07-17" "2013-08-17" "2013-09-17"
 [6] "2013-10-17" "2013-11-17" "2013-12-17" "2014-01-17" "2014-02-17"
[11] "2014-03-17" "2014-04-17" "2014-05-17" "2014-06-17" "2014-07-17"
[16] "2014-08-17" "2014-09-17" "2014-10-17" "2014-11-17" "2014-12-17"
[21] "2015-01-17" "2015-02-17" "2015-03-17" "2015-04-17" "2015-05-17"
[26] "2015-06-17" "2015-07-17" "2015-08-17" "2015-09-17" "2015-10-17"
[31] "2015-11-17" "2015-12-17" "2016-01-17" "2016-02-17" "2016-03-17"
[36] "2016-04-17" "2016-05-17"
```

TODO:

- 시간대의 영향을 보여주는 예제가 있으면 좋을 것 같음