R Data Import/Export
Version 2.9.1 (2009-06-26)

1. Introduction

분석을 위해 통계 시스템으로 데이터를 불러들이는 것(Importing)과 보고서 작성을 위해 다른 시스템으로 결과들을 내보내는 것(Exporting)은 대부분의 독자들에게 훨씬 더 흥미로운 부분이지만 통계적 분석 그 자체 보다 더 많은 시간이 걸릴 수도 있는 매우 지루한 과정이기도 합니다.
이 매뉴얼에서는 R 에 내장되어 있거나 CRAN 에서 구할 수 있는 다양한 패키지들(packages)을 통한 데이터의 입력 및 결과 출력 장치들(facilities)에 대해 기술할 것입니다. 기술된 패키지들 중 일부는 계속 개발 중이지만, 이미 충분히 유용한 기능을 제공합니다. 따로 언급하지 않는 한, 이 매뉴얼에서 기술된 모든 장치들(facilities)은 R 을 구현하는 모든 플랫폼(platform)에서 사용 가능합니다.

대체로 R 과 같은 통계 시스템들은 특히 대용량 데이터(large-scale data)의 조작에 유용합니다. 이 점에 있어서는 다른 시스템들이 R 보다 더 나을 수 있기 때문에, 이 매뉴얼의 진의 중 하나는 R 안에서 그 같은 기능을 중복해서 사용하기 보다는 우리가 다른 시스템을 사용해서 이 작업을 할 수 있다는 것을 보여주는 데 있습니다(예를 들면, Therneau & Grambsch (2000)의 저서에서 그들은 데이터 조작은 SAS 로 그리고 생존분석(survival)은 S 로 하는 것을 선호한다고 밝히기도 하였습니다!). 최근에 개발된 몇몇 패키지들은 Java, perl, python 같은 언어를 직접 R 코드와 함께 사용하는 것을 가능하게 함으로써, 이러한 언어로 만들어 낸 다양한 기능들을 더욱 유용하게 사용하고 있습니다. (Omegahat 프로젝트(http://www.omegahat.org)에서 개발된 SJava, RSPerl, RSPython 패키지와 CRAN 에서 구할 수 있는 rJava 패키지를 참조하십시오).
또한, R 도 S 처럼 작지만 다용도로 사용 가능한 여러 개의 도구(tools)를 이용하는 Unix 전통에서 출발했다는 점을 상기해 보았을 때, awk 과 perl 같은 도구를 사용하면 데이터 입력 전이나 결과 출력 후에도 데이터를 조작할 수 있도록 R 기능을 보완하는 것도 가능할 것입니다. Becker, Chambers & Wilks (1988, Chapter 9)에 있는 사례 연구에서 S 에 데이터를 입력하기 전에 여러 Unix 도구(tool)들을 데이터를 검증하고 조작하는 데 사용한 것이 그 예가 될 수 있을 것입니다. 내장된 도움말 파일 데이터베이스를 조작하기 위해 R 이 아닌 **perl** 을 이용하는 점과 함수 **read.fwf** 를 실행하는 동안(run-time) **perl** 이 필요 없다고 결정될 때까지는 자동적으로 **perl** script 를 불러낸다는 점을 볼 때, R 그 자체 역시 이러한 접근법을 택하고 있는 것으로 볼 수 있습니다. 그러한 전통적인 Unix 도구(tool)들이 이제는 Windows 를 포함한 다양한 시스템에서 더욱 많이 사용되고 있습니다.

1.1 Imports

R 에 입력하기 가장 쉬운 파일형태는 단순한 text 파일인데 이것은 주로 작거나 중간 정도 크기의 문제를 해결하는데 적합합니다. Text 파일을 읽어 들이기 위해 가장 많이 사용하는 기능은 **scan** 이며, 이것은 [Spreadsheet-likedata], Chapter2, page 5 에 설명되어 있는 더욱 편리한 대부분의 함수(functions)의 기초가 됩니다.

하지만, 모든 통계 컨설턴트들은 의뢰인이 Excel 스프레드 시트나 SPSS 파일처럼 특정 binary 포맷을 가진 데이터를 플로피 디스크나 CD 에 저장한 것을 받는 경우가 대부분입니다. 일을 하는데 있어서 흔히 가장 쉬운 경우가 데이터를 text 파일 형태로 내보내는 application 을 사용해서 그 결과물을 출력하는 것입니다(그리고 통계 컨설턴트들은 이러한 작업을 하기 위해 널리 통용되고 있는 여러 개의 application 들을 그들의 컴퓨터 안에 두고 사용하기도 합니다). 그렇지만, 이렇게 application 을 이용하는 것이 언제나 가능한 것은 아니며, [Importing from other statistical systems], Chapter 3, page 11 에서 어떠한 장치들이 R 에서 이러한 파일들을 직접 사용하는 것이 가능한지 설명하고 있습니다. Excel 스프레드 시트(spreadsheet)를 이용하기 위해 사용 가능한 방법들은 [Reading Excel spreadsheets], Chapter 8, page 26 에 요약되어 있습니다.

적지 않은 경우 데이터는 그 크기를 줄이고(compactness) 접근 속도를 높이기 위해(speed of access) binary 포맷으로 저장됩니다. 여기에 대한 한 가지 사례는 이미지 데이터(imaging data)를 저장하는 데서 찾을 수 있는데, 보통 엄청난 사이즈의 메모리를 차지하기 때문에 헤더(header)를 앞에 붙이기도 합니다. 이러한 형태의 데이터 포맷에 대해서는 [Binary files], Chapter 5, page 19 과 [Binary connections], Section 6.5, page 22 에서 더 찾아볼 수 있습니다.
좀 더 큰 데이터베이스에 대해서는 database management system (DBMS)을 사용해서 데이터를 처리하는 것이 일반적입니다. 또한, DBMS 를 사용해서 특정한 포맷이 없는(plain) 파일을 추출해내는 것도 가능합니다. 그렇지만 DBMS 와 같은 시스템들을 이용한 결과 추출 연산(extraction operation)도 R 패키지를 사용해서 직접하는 것도 가능합니다. 이에 관하여는 [Relational databases], Chapter 4 , page 13 를 참고하십시오. 네트워크 연결을 통해 데이터를 입력하는 방법은 [Network interfaces], Chapter 7, page 24 에서 찾을 수 있습니다.

1.2 Export to text files

R 에서 결과를 내보내는 과정은 대체로 다소 단순한 작업이지만, 여전히 많은 문제점을 안고 있습니다. 아무래도 이 작업에 사용하고자 하는 특정 어플리케이션이 있기 마련이고, 보통의 경우 text 파일이 가장 편리하게 변환할 수 있는 수단이라고 할 수 있습니다(만약 binary 파일이 필요하면, [Binary files], Chapter 5, page 19 을 참조하십시오).

함수 **cat** 은 데이터를 추출해내는 여러 함수들 중 가장 기본이 됩니다. 이것은 하나의 파일을 인수(argument)로 하며, 인수 **append** 는 **cat** 어떤 텍스트 파일이 연속적인 호출(call)을 통해 **cat** 에 쓰여질 수 있도록 해줍니다. 만약 이런 작업이 여러 번에 걸쳐 발생한다면, **file** connection 을 열어 쓰기(writing), 첨가하기(appending), 그리고 **cat** 을 connection 에 연결하고 **close** 로 file 을 종료하면 됩니다

가장 흔히 하는 작업은 하나의 행렬 혹은 데이터 프레임을 숫자들의 열이나 행에 label 을 붙인 사각형의 격자(grid) 형태로 입력하는 것입니다. 이 작업은 **write.table** 이나 **write** 기능들로 실행될 수 있습니다. 기능 **write** 은 단순히 정해진 숫자의 열들로 하나의 행렬이나 벡터(그리고 전치 행렬(transpose) 까지도)를 표현할 수 있습니다. 기능 **write.table** 은 더욱 사용하기 편리하며, 행과 열의 이름과 함께 데이터 프레임(혹은 데이터 프레임이라고 정의할 수 있는 하나의 개체도)을 정의할 수 있습니다.

하나의 데이터 프레임을 하나의 text 파일로 정의하기 위해서는 많은 부분에서 고민해 볼 필요가 있습니다.

1. Precision

대부분의 경우, 이러한 기능들을 이용해서 실수 복소수 간 전환(conversion)을 할 때는 가능한 모든 자릿수를 다 이용하지만(full precision), 이러한 결과 출력은 전적으로 현재 세팅 된 자릿수(digits

option)에 의해 결정됩니다. 좀 더 정확한 결과를 원한다면, 데이터 프레임에 열 별로 포맷을 지정하는 방법도 있습니다.

2. Header line

R 은 행 이름을 입력하지 않아도 되도록 header line 을 사용하는 방식을 택하며, 그렇기 때문에 파일의 형태는 다음과 같습니다.

```
                dist  climb  time
Greenmantle   2.5    650    16.083
     ...
```

몇몇 다른 시스템들은 (입력하지 않아도 되긴 하지만) 행 이름을 입력하도록 하며, 이것은 인수가 **col.names = NA** 라고 지정되어 있을 때, **write.table** 이 보여주는 결과와 같습니다.

3. Separator

가장 많이 사용되는 변수(field) 구분자 중 하나는 쉼표(comma)이며, 그래서 영어를 사용하는 국가에서는 변수 안에서 입력된 값에서 쉼표를 발견하는 일이 그리 흔하지는 않습니다. 이러한 형태의 파일들을 CSV (comma separated values)라고 하며, 일종의 포장(wrapper) 기능인 **write.csv** 가 적당한 디폴트로 작업합니다. 몇몇 지역에서는, 쉼표가 소수점(decimal point)으로 사용되고 있으므로(이 경우는 **write.table** 에서 **dec = ","** 을 지정하는 것과 같다) CSV 파일 중에 세미콜론이 변수 구분자로 사용되는 것도 있습니다. 이 경우 제대로 디폴트를 지정하기 위해 **write.csv2** 을 사용해야 합니다. 따라서, 세미콜론을 사용하거나 탭(**tap**, **sep = "\t"**)을 사용하는 것이 가장 안정적인 선택이 될 수 있습니다.

4. Missing values

디폴트로 결측값들을 **NA** 로 출력(output)되게 지정한다고 해도, 이것은 인수 **na** 에 의해 달라질 수 있습니다. **NaN** 는 **write.table** 에서는 **NA** 와 똑같이 취급되지만, **cat** 이나 **write** 에서는 그렇지 않다는 점을 주의해야 합니다.

5. Quoting strings

문자열(string, 그것이 행 이름이던 열 이름이던)은 디폴트에 의해 인용(quote)됩니다. 인수 인용은 문자(character)나 요인(factor) 변수들을 인용하는 방법을 조정하는 것 입니다. 만약에 문자열 안에 인용들을 또 포함한 경우라면 특별히 더 조심해야 합니다. 가장 사용하기 좋은 세 가지 형태는 다음과 같습니다.

```
> df <- data.frame(a = I("a \" quote"))
> write.table(df)
"a"
"1" "a \" quote"
```

```
> write.table(df, qmethod = "double")
"a"
"1" "a "" quote"
> write.table(df, quote = FALSE, sep = ",")
a
1,a " quote
```

두번째 방법은 명령을 중단(escape)하기 위한 형태인데 주로 스프레드 시트 형태 에서 많이 사용됩니다.

패키지 **MASS** 에 포함되어 있는 함수 **write.matrix** 은 행렬을 작성하는데 특화된 인터페이스를 제공하며, 행렬을 블록(block)으로 쓸 수 있는 옵션을 허용해서 메모리 사용량을 줄여줍니다.

sink 를 사용해서 파일 하나로 출력된 표준 형태의 R 결과물을 다른 형태로 전환하는 것이 가능하며, 그로 인해 (내부적으로 작동되는) **print** 에 의한 결과까지 담아냅니다. 이것이 대체로 가장 효율적인 과정(route)은 아니기 때문에 **options(width)**에 대한 세팅(setting)을 더 많이 해야 할 수 있습니다.

패키지 foreign 안에 든 기능 **write.foreign** 은 **write.table** 이라고 써서 text 파일을 생성하며 또한 다른 통계 패키지에서 text 파일로 읽을 수 있는 코드 파일도 생성한다. 현재 SPSS 와 Stata 로 결과를 내보내는 것도 가능하다.

1.3 XML

text 파일들에서 데이터를 읽어 들일 때, 그 파일을 생성하는 데 사용한 규정(converntion)들 (예를 들어 [Export to text files], Section 1.2, page 3 에 기술되어 있는 것으로 주석 기호(comment character)나 헤더의 존재 여부, 구분자, 결측치의 표현 등)을 알고 표기하는 것은 사용자의 책임입니다. 데이터의 내용뿐 아니라 그 내용의 구조까지 기록하는 마크업 언어(markup language)는 하나의 파일이 그 안에서 완전히 설명 가능하도록(self-describing) 해주며, 그렇기 때문에 누군가가 데이터를 읽을 때 소프트웨어에 이러한 세부 사항들을 알려줄 필요가 없습니다. 확장형 마크업 언어(좀 더 일반적으로는 XML 로 알려져 있는 eXtensible Markup Language)는 표준 형태의 데이터 세트뿐 아니라 좀 더 복잡한 형태의 데이터 구조까지도 표현이 가능합니다. XML 은 점점 더 엄청난 인기를 얻고 있으며, 모든 형태의 데이터 마크업과 그 교환에 대한 하나의 기준이 되어가고 있습니다. XML 은 지도와 그래픽을 표현하는데 그리고 수학식 등을 표현하는데 사용되고 있습니다.

패키지 XML 은 R 과 S-PLUS 모두에서 XML 문서를 읽고 쓸 수 있도록 다양한 기능을 제공하여, 기술의 진화에 따라 우리가 좀 더 쉽게 이 기술을 사용할 수도 있다는 희망을 실현하고자 합니다. 몇몇 사람들은 우리가 XML 을 이용하여 다른 기술과 다른 무엇을 할 수 있는지 탐구 중 입니다.

데이터 셋을 다양한 application 들에서 공통으로 사용할 수 있도록 표현하는 것, R 과 S-PLUS 에 저장된 개체들은 두 개의 시스템 모두에서 공유 가능하게 하는 것, 플롯은 SVG(Scalable Vector Graphics, 일종의 XML)을 통해서 표현하는 것, 기능을 문서화 하는 것, text 와 데이터 그리고 코드를 모두 포함하는 "살아있는" 분석 또는 보고서를 작성하는 것 등이 그것입니다.

패키지 XML 의 다양한 기능들에 설명은 이 문서의 설명 범위를 벗어납니다. 좀 더 자세한 내용과 예시가 필요하면 web page 를 찾아보십시오(http://www.omegahat.org/RSXML). CRAN 에서 찾을 수 있는 패키지 StatDataML 은 XML 로 작성된 하나의 예시입니다.

2. Spreadsheet-like data

[Export to text files], Section 1.2, page 3 에서 우리는 행과 열 이름이 달려 있고 사각의 격자 형태로 표현된 다양한 스프레드 시트 형태의 text 파일 포맷을 확인할 수 있었습니다. 이번에는 이와 같은 형태의 파일들을 R 로 읽어 들이는 방법을 다룹니다.

2.1 Variations on read.table

read.table 은 사각형 격자 형태의 데이터를 읽기 위한 가장 편리한 방법입니다. 다양한 가능성 때문에, **read.table** 기능을 하는 다른 다양한 기능들이 존재하지만, 이러한 기능들은 단지 몇 개의 디폴트 지정만 달리한 것입니다. 주의할 점은 **read.table** 은 대용량 수치 행렬을 읽기에는 비효율적인 방법이라는 점입니다. 보다 자세한 내용은 다음에 나올 **scan** 을 참고하십시오.

몇 개의 중요한 사항은 다음과 같습니다.

1. Encoding

만약 파일에 ASCII 형태가 아닌 문자 변수들이 포함되어 있다면, 그것이 제대로 부호화 되어 읽어졌는지 반드시 확인해야 합니다. 이것은 주로 UTF-8 locale 에서 Latin-1 파일을 읽는 것과 관련된 문제인데, 이것은 아래와 같은 방법으로 해결할 수 있습니다.

read.table(file("file.dat", encoding="latin1"))

중요한 것은 Latin-1 으로 표현된 모든 locale 에 대해 이 방법을 사용할 수 있다는 점입니다.

2. Header line

우리는 여러분들에게 헤더를 분명하게 표기하라고 권하고 싶습니다. 전통적으로 헤더에는 행 이름 대신 열 이름들만 들어가 있었는데, 그래서 하나의 변수가 다른 것들에 비해 길이가 짧습니다(만약 R 이 이것을 판단할 수 있으면, **header = TRUE** 로 세팅 할 것입니다). 만약에, 어떤

파일이 행 이름에 대한(입력이 되지 않았을 수 있는) 헤더 변수를 포함한다면, 아래와 같은 방법으로 이 파일을 읽을 수 있습니다.

read.table("file.dat", header = TRUE, row.names = 1)

열 이름들은 **col.names** 을 사용해서 따로 정의될 수 있으며, 따로 정의된 이름은 헤더(만약 존재한다면)에 대해 우선합니다.

3. Separator

보통은 파일을 보면 파일에 사용된 변수 구분자가 무엇인지 알 수 있지만, 공백으로 파일이 구분된 경우, 우리는 구분자로 어떤 종류의 공백이든(단순 여백, 탭, 혹은 새로운 행) 다 사용할 수 있는 디폴트 **sep = ""**과 **sep = " "**, **sep = "\t"** 중에서 하나를 구분자로 선택할 수 있습니다. 구분자의 선택이 인용된 문자열의 입력에 영향을 줄 수도 있다는 점을 주의해야 합니다. 만약 당신이 여백을 포함하고 있는 탭을 구분자로 지정한(tab-delimited) 파일을 가지고 있다면, **sep = "\t"**을 사용해야만 합니다.

4. Quoting

디폴트에 의해 문자형 문자열이 '"' 또는 '’'에 의해 인용되면, 각각의 경우에 그 인용부호(quote)가 나타나기 전의 모든 문자들이 문자형 문자열의 한 덩어리로 간주됩니다. 유효한 인용부호 문자들의 조합(만약 인용부호 지정이 없다면)은 **quote** 인수(argument)로 통제할 수 있습니다. **sep ="\n"**에 대해서, 디폴트는 **quote = ""**로 변경되어야 합니다.

아무런 구분자가 지정되지 않은 경우, C 에서 그런 것처럼, 인용된 문자열 안에 든 인접한 바로 앞의 '\'에 의해 인용 작업이 중단될 수 있습니다. 특정 구분 문자가 지정된 경우에는, 스프레드 시트에서 많이 발생하는 것처럼, 인용된 문자열 내에서 중복된 구분 문자에 의한 인용 작업의 중단이 발생할 수도 있습니다. 예를 들면, 'One string isn''t two', "one more" 을 읽어 들일 때는 **read.table("testfile", sep = ",")**을 사용해야 합니다. 이렇게 하면 디폴트 구분자에 대해 작동하지 않습니다.

5. Missing values
파일 안에 든 결측값들을 표기하기 위해 문자열 **NA** 를 사용하도록 디폴트가 지정되어 있지만, 결측값을 표현하기 위한 한 개 이상의 문자 벡터를 지정하는 **na.strings** 에 의해 이 표현은 달라질 수 있습니다. 또한 수치화된 열들 중에서 비워진 열들도 결측치로 간주됩니다. 숫자로 된 열에서는 **NaN** 과 **Inf**, **-Inf** 이 값 채워진 것으로 인정됩니다.

6. Unfilled lines
스프레드 시트로부터 나온 파일이 줄줄 이어지는 빈 칸이 있는 열 들을 포함했거나 구분자가 빠져있는 경우는 매우 흔합니다. 이러한 파일을 읽기 위해서는 **fill = TRUE** 옵션을 사용합니다.

7. White space in character fields
구분자가 지정되어 있는 경우에는, 문자열들 앞에 나오는 계속되는 흰 공간이 그 열의 일부로 인식됩니다. 그 공간을 벗겨내기 위해서는, **strip.white = TRUE** 옵션을 사용합니다.

8. Blank lines
read.table 은 디폴트에 의해 공백인 행들을 무시합니다. 이것은 **blank.lines.skip = FALSE** 세팅에 의해 바뀔 수 있는데, 이 옵션은 **fill = TRUE** 과 함께 사용될 때만 작동합니다. 이러한 현상은 아마도 보통의 layout 에서는 비어 있는 행들이 결측된 개체들을 의미하기 때문일 것 입니다.

9. Classes for the variables
특별한 조치를 따로 하지 않는다면, **read.table** 은 데이터 프레임 내에 포함된 각각의 변수들에 적당한 class 를 스스로 할당하려 합니다. 어떤 입력 값이 결측치가 아니고 특별히 해당 class 로 전환 될 수 없다면, 이 문장은 차례로 **logical**, **integer**, **numeric** 그리고 **complex** 로 움직여가며 class 할당을 시도합니다.[1] 만약 이러한 시도가 모두 실패하면, 그 변수는 factor 변수로 변환됩니다.

인수(argument) **colClasses** 와 **as.is** 는 더 많은 것을 통제할 수 있게 합니다. **as.is** 는 문자열들이 factor 변수로 전환되는 일이 없도록 통제합니다(주로 이 용도로만 쓰입니다). **colClasses** 를 사용하면, 입력 시 각각의 열에 전부 다른 class 를 할당하는 것이 가능합니다. 중요한 것은 **colClasses** 와 **as.is** 이 각각의 변수가 아니라, 각각의 열에 대해 기술한다는 것입니다. 그래서 이들은 행 이름들이 포함된 열도 (만약 그런 열이 있다면) 포함합니다.

10. Comments
read.table 은 디폴트로 '#'를 주석 문자로 사용하고 있습니다. 만약, 이 문자가 나타난다면 (인용되고 있는 부분에서 나타난 것이 아니라면) 그 줄의 나머지 부분은 무시됩니다. 즉, 오직 흰 공간과 주석을 포함한 줄들 만이 공백으로 처리됩니다. 만약 데이터 파일 안에 주석이 없는 것이 확실하다면, **comment.char = ""**을 사용하는 편이 더 안전할 것입니다(어쩌면 처리도 더 빠를 것입니다.)

11. Escapes
많은 OS 들은 관습적으로 text 파일에서 역슬래쉬(\)를 마침(또는 종료: escape) 문자로 사용하고 있지만, Windows 에서는 그렇지 않습니다(역슬래쉬(\)를 경로 이름을 표시하는 데 사용하고 있습니다). R 에서 이 문제는 데이터 파일에 적용하는데 있어 관습을 따를 것인지 아닌지의 선택의 문제입니다.

read.table 과 **scan** 에서는 모두 **allowEscapes** 라는 논리 인수(logical argument)를 사용합니다. R 2.2.0 버전부터는 이 인수가 false 가 디폴트로 설정되어 있으며, 그래서 (앞서 설명한 환경에서는) 역슬래쉬(\)들은 오직 마침(또는 종료: escaping) 인용문으로 해석됩니다.

[1] 이 방법은 보통 첫 번째 입력 값에서 다른 가능성을 배제하는 방법만큼이나 빠릅니다.

read.table 과 **scan** 둘 모두 논리 인수(logical argument) **allowEscapes** 를 가지고 있습니다. R 2.2.0 에서부터와 같이 이것은 false 가 디폴트로 지정되어 있으며 (위에서 설명한 환경하에서) 역슬래쉬(\)는 단지 인용을 마치는 것으로 해석됩니다. 만약 이러한 세팅이 true 라면, C 스타일로 \a, \b, \f, \n, \r, \t, \v 와 \040 and \0x2A 와 같이 표현된 8 진수나 16 진수 통제 문자가 종료로 해석됩니다. 역슬래쉬(\)를 포함한 어떤 다른 종료 문자도 그것과 같이 다루어 집니다. **read.csv** 와 **read.delim** 와 같은 편리한 함수는 영문사용 부분에서 스프레드시트에서 내보내진 CSV 파일과 탭으로 분리된 파일에 적절하게 read.table 에 인수를 제공합니다. **read.csv2** 와 **read.delim2** 는 소수점으로 콤마(comma)를 사용하는 국가에 적당합니다.

read.table 에 옵션을 부적절하게 지정하면, 보통 다음과 같은 형태의 에러 메시지가 뜹니다.

Error in scan(file = file, what = what, sep = sep, :
line 1 did not have 5 elements

또는

Error in read.table("files.dat", header = TRUE) :
more columns than column names

이 메세지가 무엇이 문제인지 충분한 정보를 제공하지 못한다면 **count.fields** 가 문제를 보다 더 자세하게 찾아내는데 유용하게 사용될 수 있습니다.

효율성은 큰 데이터 그리드(grid)를 읽는데 중요합니다. 각 열에 대하여 (논리, 정수, 분수, 복소수, 문자 또는 미가공 된) atomic 벡터 유형들 중 하나인 **colClasses** 인 **comment.char = ""**를 특정하고 읽어야 할 행의 수인 **nrows** 를 부여하는데 (그리고 다소 과대 추정하는 것이 특정하는 것보다 더 좋습니다) 도움이 될 것입니다. 아래의 예제들을 참조하십시오.

2.2 Fixed-width-format files

가끔 데이터 파일은 필드 구분자가 없지만 사전에 정해진 열에 필드를 가집니다. 이것은 천공카드가 사용되던 때에 매우 일반적이었고 파일 공간을 절약하는데 여전히 때때로 쓰이기도 합니다.

함수 **read.fwf** 는 벡터의 필드 넓이를 특정지음으로써, 그러한 파일들을 읽는데 단순한 방법을 제공합니다. 그 함수는 파일의 전체 줄을 메모리로 읽고, 결과 문자열을 분할하고, 탭으로 분리된 임시 파일을 완성하고, 그리고 **read.table** 을 호출(call)합니다. 이것은 작은 파일에 적당하지만 어떤 더 복잡한 것에 대해서는 사전 작업된 파일을 위해 **perl** 과 같은 언어도구를 사용하는 것을 추천합니다.

함수 **read.fortran** 은 포트란(Fortran) 스타일의 열을 지정함으로써 고정된 포맷 파일에 대한 유사한 함수입니다.

2.3 Data Interchange Format (DIF)

스프레드시트와 같은 데이터에 사용된 어떤 특이한 포맷은 DIF(Data Interchange Format)입니다. 함수 **read.DIF** 은 이러한 파일은 읽는데 단순한 방법을 제공합니다. 이것은 각 열에 유형을 할당하는데 **read.table** 과 유사한 인수(argument)를 취합니다.
윈도우에서 스프레드시트들은 종종 이 포맷에서 클립보드의 스프레드시트 데이터를 저장합니다. **read.DIF**("**clipboard**")는 직접 그곳으로부터 데이터를 읽을 수 있습니다. 이것은 빈 셀이 있는 스프레드시트를 다루는데 있어서 **read.table**("**clipboard**")보다 약간 더 강력합니다.

2.4 Using scan directly

read.table 와 **read.fwf** 둘은 파일을 읽는데 **scan** 을 이용하고 **scan** 의 결과를 처리합니다. 그것들은 매우 편리하지만 종종 **scan** 을 직접 사용하는 것이 더 낳을 때도 있습니다.
함수 scan 은 많은 인수(argument)를 가지고 있고 그것들 중 대부분은 이미 **read.table** 에 속해있습니다. 가장 중요한 인수(argument)는 **what** 으로 이것은 어느 것이 파일로부터 읽혀질 변수의 모드(mode) 목록인지 특정하는 것입니다. 만약 이 목록에 이름이 붙여져 있으면, 그 이름은 반환되는 목록의 성분을 위해 사용됩니다. 모드(mode)는 수, 문자 또는 복소수가 될 수 있고, 보통 **0**, **“”** or **0i** 와 같은 사례에 의해 특정됩니다. 예를 들면 아래와 같습니다.

```
cat("2 3 5 7", "11 13 17 19", file="ex.dat", sep="\n")
scan(file="ex.dat", what=list(x=0, y="", z=0), flush=TRUE)
```

이것은 세 가지 구성요소와 목록을 출력하고 파일의 네 번째 열을 버립니다.

readLines 라는 함수는 원하는 모든 것이 앞으로의 작업진행을 위해 모든 라인(line)을 읽는 것이라면 보다 편리할 것입니다.

scan 의 일반적인 용법 중 하나는 큰 행렬의 데이터를 읽는 것입니다. 행과 열이 각각 200 과 2000 인 행렬 데이터 matrix.dat 이 있다고 가정하고 이것을 읽어 들이는데 다음과 같이 하면 유닉스 체제에서는 1 초 이하(같은 컴퓨터에서 윈도우 체제를 사용하면 3 초 이하)의 시간 밖에 걸리지 않습니다.

```
A <- matrix(scan("matrix.dat", n = 200*2000), 200, 2000, byrow = TRUE)
```

반면, 아래와 같이 하면 더 많은 메모리와 시간(10 초)이 걸리고 마지막 세 번째 방법으로 하면 7 초가 걸립니다.

A <- as.matrix(read.table("matrix.dat"))

A <- as.matrix(read.table("matrix.dat", header = FALSE, nrows = 200,comment.char = "", colClasses = "numeric"))

이 같은 차이는 거의 모두 2000 개의 분리된 짧은 열 전체 때문입니다. 그것들의 길이가 200 이면, **scan** 은 9 초가 걸리는 반면 **read.table** 은 특별히 **colClasses** 를 사용하여 효율적으로 사용한 경우 18 초, 특별한 처리 없이 그냥 사용할 경우 125 초가 소요됩니다.

결국 소요되는 시간은 읽기 유형과 데이터에 따라서 달라질 수 있음에 주의하십시오. 백만개의 별개의 정수를 읽는 것을 가정하여 보면 다음과 같습니다.

```
writeLines(as.character((1+1e6):2e6), "ints.dat")
xi <- scan("ints.dat", what=integer(0), n=1e6)          # 0.77s
xn <- scan("ints.dat", what=numeric(0), n=1e6)          # 0.93s
xc <- scan("ints.dat", what=character(0), n=1e6)        # 0.85s
xf <- as.factor(xc)                                     # 2.2s
DF <- read.table("ints.dat")                            # 4.5s
```

그리고 백만개의 작은 코드 집합의 예는 다음과 같습니다.

```
code <- c("LMH", "SJC", "CHCH", "SPC", "SOM")
writeLines(sample(code, 1e6, replace=TRUE), "code.dat")
y <- scan("code.dat", what=character(0), n=1e6)         # 0.44s
yf <- as.factor(y)                                      # 0.21s
DF <- read.table("code.dat")                            # 4.9s
DF <- read.table("code.dat", nrows=1e6)                 # 3.6s
```

이러한 소요시간은 운영체제(기본적인 읽기 기능도 윈도우 운영체제에서는 유닉스 운영체제보다 최소한 두 배의 시간이 소요됩니다)와 garbage collector(불필요한 정보를 정리하여 정리된 공간을 만드는 것)의 정확한 상태에 따라 영향을 매우 많이 받습니다.

2.5 Re-shaping data

때때로 스프레드시트 형태의 데이터는 목적에 따른 모든 관측치들에 의해 그 각각의 목적에 따른 공변량을 부여하는 단순한 포맷으로 되어 있습니다. R 의 모델링 함수들은 하나의 열에 있는 관측치를 필요로 합니다. 다음과 같은 반복적으로 측정한 뇌 MRI 측정값으로부터 얻은 표본 데이터를 고려해 보십시오.

Status Age V1 V2 V3 V4
P 23646 45190 50333 55166 56271
CC 26174 35535 38227 37911 41184
CC 27723 25691 25712 26144 26398
CC 27193 30949 29693 29754 30772
CC 24370 50542 51966 54341 54273
CC 28359 58591 58803 59435 61292
CC 25136 45801 45389 47197 47126

여기엔 두 개의 공변량이 있고 각각의 목적에 네 개의 관측치가 추가되어 있습니다. 이 데이터는 **mr.csv** 과 같은 엑셀파일로부터 읽어 들인 것입니다. 다음과 같이 **stack** 을 이용하면 데이터를 단순한 형태의 출력결과를 보일 수 있도록 하는데 도움이 됩니다.

```
zz <- read.csv("mr.csv", strip.white = TRUE)
zzz <- cbind(zz[gl(nrow(zz), 1, 4*nrow(zz)), 1:2], stack(zz[, 3:6]))
```

이의 출력형태는 아래와 같습니다.

Status Age values ind
X1 P 23646 45190 V1
X2 CC 26174 35535 V1
X3 CC 27723 25691 V1
X4 CC 27193 30949 V1
X5 CC 24370 50542 V1
X6 CC 28359 58591 V1
X7 CC 25136 45801 V1
X11 P 23646 50333 V2
...

함수 **unstack** 는 반대 방향으로 작동하는데 데이터 내보내기에 유용합니다. 이러한 작업을 하는 다른 방법은 아래와 같이 함수 **reshape** 를 사용하는 것입니다.

```
> reshape(zz, idvar="id",timevar="var",
varying=list(c("V1","V2","V3","V4")),direction="long")
```
Status Age var V1 id
1.1 P 23646 1 45190 1
2.1 CC 26174 1 35535 2
3.1 CC 27723 1 25691 3
4.1 CC 27193 1 30949 4
5.1 CC 24370 1 50542 5
6.1 CC 28359 1 58591 6

7.1 CC 25136 1 45801 7
1.2 P 23646 2 50333 1
2.2 CC 26174 2 38227 2
...

reshape 는 **stack** 보다 더 복잡한 구문을 사용해야 하지만 사례에서의 한 열보다 더 긴 형태를 가지는 데이터를 다루는 데 사용될 수 있습니다. direction="wide"과 함께 reshape 또한 반대 방향으로 데이터를 전환하는 기능을 수행할 수 있습니다.

2.6 Flat contingency tables

다차원의 수렴된 표를 표현하기 위해서 배열의 형태는 오히려 불편합니다. 범주형 데이터 분석의 경우 그러한 정보는 종종 셀의 수에 따라 factor 의 수준 조합을 특정하여 첫번째 열과 행과 함께 확장된 이차원 배열로 표현합니다. 행은 위에서부터 아래로 읽고 열은 왼쪽에서 오른쪽으로 읽는 명백한 관습처럼 이러한 행과 열들은 라벨이 단지 그것이 변할 때만 표시된다는 의미에서 구식입니다(ragged). R 에서 그러한 평범한 형태의 수렴된 표는 **ftable** 를 이용하여 생성할 수 있으며 그것은 적절한 출력 방법과 함께 "**ftable**" 클래스 대상을 생성합니다.

UCBAdmissions 이라는 R 의 표준적인 데이터 셋을 예로 하나 들어보면, 입학허가와 성별로 구분된 1973 년 가장 큰 여섯 부문에 대한 UC Berkeley 대학원에 지원한 지원자들을 분류한 결과를 담고 있는 3 차원의 수렴된 표를 고려해 볼 수 있습니다.

> data(UCBAdmissions)
> ftable(UCBAdmissions)
Dept A B C D E F
Admit Gender
Admitted Male 512 353 120 138 53 22
Female 89 17 202 131 94 24
Rejected Male 313 207 205 279 138 351
Female 19 8 391 244 299 317

출력된 모양은 명백히 3 차원 배열로 나타내어진 데이터보다 더 유용합니다.

파일에 있는 평범한 표(flat-like contingency tables)를 읽기위한 read.ftable 이라는 함수도 있습니다.

이 함수는 행과 열이 담고있는 변수명과 값을 표현하는데 있어 표가 담고있는 다양한 정보를 정확하게 다루는데 사용되는 추가적인 인수(argument)도 추가적으로 가지고 있습니다. read.ftable 을 위한 도움말은 다수의 유용한 예들을 가지고 있습니다. flat.table 은 as.table 을 사용하여 배열형태(array form)에서 standard contingency table 로 전환할 수 있습니다.

flat.table 은 그것의 행(또는 열)의 "ragged" display 로 특징지어집니다. 만약 행 변수값의 모든 격자(full grid)가 주어진다면 어떤 사람은 데이터를 읽기 위해 read.table 을 대신 사용하고 xtabs 를 사용하여 이것으로부터 contingency table 을 생성하여야 합니다.

3 Importing from other statistical systems

이 장에서는 다른 통계시스템에 의해 작성된 바이너리 데이터 파일을 읽는데 따르는 문제에 대하여 설명합니다.

이것은 대부분의 경우 문제를 피하는 가장 좋은 방법이지만 기본 시스템이 지원하지 않을 경우 해결되지 않을 수도 있습니다.

3.1 EpiInfo, Minitab, S-PLUS, SAS, SPSS, Stata, Systat

추천할만한 다른 외부의 패키지는 이런 통계시스템에 의해 마들어진 파일을 불러오거나 내보내는 기능을 제공합니다.

몇몇의 경우 이러한 함수는 read.table 이 필요로 하는 것에 비해 실질적으로 적은 메모리를 필요로 할 것입니다. write.foreign 은 현재 SPSS 와 STATA 로 내보내기 기능을 제공합니다.

EpiInfo version 5 와 6 은 자체적으로 고정된 길이의 텍스트 포맷으로 데이터를 저장했습니다.

read.epiinfo 는 '.REC' 류의 파일을 R 데이터 프레임으로 읽어들일 것입니다. EpiData 또한 이러한 포맷의 데이터를 생성합니다.

함수 read.mtp 는 Minitab Portable Worksheet 를 불러오기 합니다. 이것은 R list 처럼 워크시트의 구상요소를 반환합니다.

함수 read.xport 는 SAS Transport (XPORT) 포맷의 파일을 읽고 데이터 프레임 목록(a list of data frame)을 반환합니다. 만약 SAS 가 당신의 컴퓨터에서 사용가능하다면, 함수 read.ssd 가 변환포맷에서(in Transport format) SAS 영구 데이터셋('.ssd' 또는 '.sas7bdat')으로 저장되어 있는 SAS 스크립트를 생성하고 실행하는데 사용될 수 있습니다.

그러고 난 후 결과 파일을 읽기위해 read.xport 를 호출할 수 있습니다. (패키지 Hmisc 는 유사한 함수인 sas.get 을 가지고 있으며 SAS 를 실행합니다.) SAS 접속없이 윈도우즈에서 실행하는 경우 무료로 구할 수 있는 SAS System Viewer 는 SAS 데이터셋을 열고 그것을 '.csv'와 같은 포맷으로 내보내기하는데 사용할 수 있습니다.

함수 read.S 는 S-PLUS 3.x, 4.x 또는 Unix 나 Windows 2000 (32-bit)에 의해 생성된 바이너리 object 들을 읽을 수 있고 그것들을 다른 OS 에서도 읽을 수 있습니다. 이것은 많이 읽을 수 있는데다 모두 S object 일 필요도 없습니다. 특히 그것은 벡터, 행렬 그리고 테이터 프레임과 그것들을 담고있는 list 들도 읽을 수 있습니다.

함수 data.restore 는 (Alpha 플랫폼에서 dump 된 것을 제외하면 또한 읽을 수 있습니다) (data.dump 에 의해 생성된) S-PLUS 데이터 덤프 (dara dump) 같은 제약하에 읽어 들입니다. 그것은 S-PLUS 5.x 과 그 이후의 버전에서 data.dump(oldStyle=T)로 생성된 데이터 덤프를 읽도록 될 것입니다.

만약 당신이 S-PLUS 에 접근할 수 있다면, S-PLUS 에서 object(들)과 R 에서 생성된 덤프파일 소스(source)를 덤프(dump)하는 것이 보통의 경우 보다 더 신뢰할만 합니다. S-PLUS 5.x 와 6.x 의 경우 당신은 아마도 dump(..., oldStyle=T)를 사용할 필요가 생길 것이고 매우 큰 object 를 읽으면 소스함수(source function)의 사용보다 배치 스크립트(batch script)로서 덤프파일을 사용하는 것을 선호한할 수 있습니다.

함수 read.spas 는 SPSS 에서 '저장하기(save)'와 '내보내기(export)' 명령문에 의해 생성된 파일을 읽을 수 있습니다. 그리고 저장된 데이터셋의 각 변수에 따른 하나의 구성요소와 함께 그 목록(list)를 반환합니다. SPSS 의 변수명(value label)변수들은 선택적으로 R 요소(R factor)로 변환됩니다.

SPSS Data Entry 는 데이터 엔트리 형태(entry forms)를 생성하는 어플리케이션의 하나입니다. 디폴트로 read.spss 는 다룰 수 없지만 원래의 SPSS 포맷의 데이터를 추가적 포맷 정보를 담아 생성할 수 있습니다.

Stata '.dta'파일은 바이너리 포맷의 파일 중 하나입니다. Stata 의 버전 5, 6, 7/SE 그리고 8 의 파일은 함수 read.dta 와 write.dta 에 의해 읽고 쓸 수 이씁니다. Stata 변수는 변수명과 함께 R 요소 (R factor)로 변환시키거나 R 요소 (R factor)로부터 변환될 수 있습니다.

read.systat 는 (Windows 와 같은) little-endian machines 에 쓰여진 장방형의 데이터 파일 (mtype=1)인 Systat SAVE 파일을 읽습니다. 이것들은 '.sys' 또는 (보다 최근에는) '.syd'의 확장자를 가지고 있습니다

3.2 Octave

Octave 는 수치선형대수시스템(numerical linear algebra system)입니다. 외부패키지 내의 함수 read.octave 는 Octave 명령어인 save-ascii 를 이용하여 생성된 Octave 텍스트 데이터 포맷의 파일을 읽을 수 있습니다. 이 함수는 일반적인 형태의 대부분의 변수, 표준 atomic(실수, 복소수, 행렬, 그리고 N 차원의 배열(array), 끈(string), 범위(range) 및 boolean 수와 행렬)과 가역적인 것들(structs, cells, 그리고 목록(lists))을 다룰 수 있습니다.

4 Relational databases

4.1 Why use a database?

R 이 잘 다룰 수 있는 데이터 형태에는 제한이 있습니다. R 에 의해 조작된 모든 데이터는 메모리 상에 존재하고 데이터의 여러 복제본이 함수를 실행하는 동안 생성될 수 있기 때문에 R 은 아주 큰 데이터셋에는 적합하지 않습니다. 크기가 몇 백 메가바이트 이상인 데이터 object 는 R 구동시 메모리 용량초과문제를 야기할 수 있습니다.

R 은 데이터 동시접속이 쉽게 지원되지 않습니다. 즉 한 명 이상의 사용자가 동시에 동일한 데이터에 접속하거나 업데이트 하면 한 사용자가 만든 변화가 다른 사용자에게 보이지 않을 수 있습니다.

R 은 영구데이터를 지원하므로 한 세션에서 데이터 object 나 전체 워크시트를 저장할 수 있고 이어지는 세션에 담을 수 있습니다. 그러나 저장된 데이터 포맷은 R 에 특유한 것으로서 다른 시스템에 의해 쉽게 조작되지 않습니다.

대이터관리시스템(DBMSs)와 특히 관련 DBMSs(RDBMSs)는 위의 모든 것들을 잘 처리할 수 있도록 고안되었습니다. 이것의 장점은 다음과 같습니다.

1. 큰 데이터베이스에서 선택한 부분에 빠르게 접근할 수 있습니다.

2. 테이터베이스에서 요약하거나 열간 교차표를 만드는 강력한 방법을 제공합니다

3. 스프레드시트의 격자모양의 장방형 모형이나 R 데이터 프레임보다 더 조직화된 형태로 데이터를 저장합니다.

4. 데이터 접속에 보안제약을 강화하면서 복수의 호스트에서 작업을 하는 다수의 클라이언트가 동시접속 할 수 있습니다.

5. 넓은 범위의 클라이어트를 지원하는 서버로서 운용할 수 있습니다.

다차원의 contingency 표를 만들어 내기 위해서 DBMS 가 쓰일 수도 있는 통계적 어플리케이션 부류들은 데이터의 10%를 교차표를 만들기 위해서 표본으로 추출할 수도 있습니다. 그리고 분리된 분석에 사용할 데이터를 데이터베이스에서 그룹별로 추출할 것입니다.

4.2 Overview of RDBMSs

전통적으로 데이터 보안에 보다 비중을 두고 있는 비싸고 큰 상업용 RDBMSs 는 있었습니다(Informix; Oracle; Sybase; IBM's DB/2; Microsoft SQL Server on Windows). 그리고 교육용이나 작은 크기의 데이터베이스(MySQL, PostgreSQL, Microsoft Access 와 같은)도 있었습니다.

오픈소스인 PostgreSQL 이 점점 고도화되고 Informix, Oracle, 그리고 Sybase 가 리눅스에서 무료 버전을 배포하면서 이러한 관점의 논의는 의미가 희미해져 가고 있습니다.

텍스트 파일뿐만 아니라 관련된 데이터베이스를 사용하는 다른 일반적인 - 스프레드시트를 포함하여 -데이터 원천들이 있습니다. ODBC(Open Database Connectivity)는 이런 종류의 모든 데이터 원천 사용의 표준입니다.

그것은 Windows 에 기반한 것이었지만 Linux/Unix 에서도 사용할 수 있습니다(http://www.microsoft.com/data/odbc 를 참고하시오). 이 장의 후반부에 클라이언트 대 글라이언트/서버 데이터베이스를 제공하는 모든 패키지가 설명되어 있습니다.

데이터베이스는 같은 machine 또는 (보다 자주) 원격적으로 같이 존재할 수 있습니다. 여러 DBMSs 를 지원하는 SQL(Structured Query Language, 종종 'sequel'이라 불리기도 합니다; Bowman et al. 1996 과 Kline 과 Kline 2001 을 참고하십시오) 인터페이스 언어에 대한 ISO 표준이 있습니다(실제로는 SQL92 는 ISO/IEC 9075 또는 ANSI X3.135-1992 라고 알려져 있고 SQL99 가 곧 사용될 것입니다).

4.2.1 SQL queries

보다 포괄적인 R 인터페이스들은 일상적인 운용 뒤의 SQL 을 생성하지만 SQL 의 직접적인 사용이 복잡한 운용에 있어 필요합니다.

전통적으로 SQL 은 대문자로 쓰여졌지만 많은 이용자들이 R 인터페이스 함수에서 소문자로 쓰는 것이 더 편리함을 느낄 것입니다.

관련된 DBMS 는 그것들이 하나의 실체에 대한 관측치를 담고 있는 열들 또는 필드의 한 종류(수치, 문자, 날짜, 등)과 행 또는 레코드로 만들어진 점에서 데이터를 R 데이터 프레임과 비슷한 표들(또는 관계들)의 데오터베이스 처럼 저장합니다.

SQL 쿼리(query)는 관련 데이터베이스에서 아주 일반적으로 운용되는 것입니다. 전통적인 쿼리는 다음과 같은 유형의 SELECT 구문입니다.

SELECT State, Murder FROM USArrests WHERE Rape > 30 ORDER BY Murder SELECT t.sch, c.meanses, t.sex, t.achieve
FROM student as t, school as c WHERE t.sch = c.id
SELECT sex, COUNT(*) FROM student GROUP BY sex
SELECT sch, AVG(sestat) FROM student GROUP BY sch LIMIT 10

이들 중 첫번째는 데이터베이스 표에 교차로 복사된 R 데이터 프레임 USArrest 로부터 두 열을 선택합니다. 그리고 세 번째 열에서 결과가 정렬되도록 합니다.

두 번째는 학생과 학교의 두 표를 네 번째 열에 반환하도록 데이터베이스 결합 기능을 수행합니다. 세 번째와 네 번째 쿼리는 교차표를 작성하고 회수와 평균을 반환합니다(다섯 개 총 함수는 COUNT(*)와 SUM, MAX, MAX, MIN 그리고 AVG 이고 각 함수는 하나의 컬럼씩에 적용됩니다).

SELECT 쿼리는 표를 선택하는데 FROM 을, 포함(또는 AND 또는 OR 로 구분된 하나 이상의 조건)에 대한 조건을 특정하기 위해서 WHERE 를, 그리고 결과를 정렬하기 위해 ORDER BY 를 사용합니다.

데이터 프레임과는 다르게 RDBMS 표에서 행들은 마치 ORDER BY 명령어 없이 순서가 정해져 있지 않은 경우들에 가장 최선의 방법입니다. 여러분은 콤마(,)로 구분하여 하나 이상의 열을 정렬(사전편찬식의 순서로)할 수 있습니다.

SELECT DISTINCT 쿼리는 선택된 표에서 각각 구분된 행의 하나의 사본을 반환합니다.

GROUP BY 구문은 기준에 따라 행들의 하위 그룹을 선택합니다. 하나 이상의 열이 콤마(,)에 의해 구분되어 특정되면 다중교차분류(multi-way cross classifications)가 다섯개의 결합함수 중 하나에 의해 요약될 수 있습니다.

HAVING 구문은 결합된 값에 따라 포함/제외 그룹을 선택학게 할 수 있습니다.

만약 SELECT 명령문이 유일한 순서를 생성하는 ORDER BY 문을 포함한다면, LIMIT 구문은 결과 행들의 연속된 묶음(a contiguous block of output rows)을 선택하여 추가할 수 있습니다. 이것은 일시에 묶음 단위의 열을 추출하는데 유용할 수 있습니다(어쩌면 순서가 유일하지 않는한 LIMIT 구문이 쿼리를 최적화하는 데 사용 가능한 것과 같이 신뢰할만 하지 않을 수도 있습니다).

CREATE TABLE 과 같이 표를 만들지만 보통은 이 인터페이스에서 데이터프레임을 데이터베이스에 복사하는 INSERT, DELETE 또는 UPDATE data 가 있습니다. 표는 쿼리 DROP TABLE 로 삭제될 수 있습니다.

Kline and Kline(2001)은 SQL 서버 2000, 오라클. MySQL 과 PostgreSQL 에서 SQL 도입에 대해 자세히 논하였습니다.

4.2.2 Data types

데이터는 다양한 형태로 데이터베이스에 저장될 수 있습니다. 데이터 형태의 범의는 DBMS 에 따라 결정되지만, (SQL 의 이름이 아니라) SQL 표준이 아래에 나열되어 있는 널리 도입되어 있는 것을 포함하여 많은 유형을 정의하고 있습니다.

float(p) Real number, with optional precision. Often called real or double or double precision.

integer 32-bit integer. Often called int.

smallint 16-bit integer

character(n) fixed-length character string. Often called char.

character varying(n) variable-length character string. Often called varchar. Almost always has a limit of 255 chars.

boolean true or false. Sometimes called bool or bit.

date calendar date

time time of day

timestamp date and time

timezone 과 함께 time 과 timestamp 에는 많은 변형이 있습니다. 대용량의 텍스트와 바이너리 데이터 블럭을 위해 도입된 다른 많은 유형들이 각각 텍스트와 벌브(text and blob)의 형태로 있습니다. 더욱 포괄적인 R 인더페이스 패키지가 사용자로부터 형변환(the type conversion)을 숨겨져 있습니다.

4.3 R interface packages

R 도움말을 위해 DBMSs 와 연계된 CRAN 에 관한 여러 종류의 패키지가 사용할 수 있습니다. 이것들은 여러 수준의 추론(abstraction)을 제공합니다. 어떤 것은 데이터베이스로부터 그리고 데이터베이스로 전체 데이터로 복사하는 기능을 제공합니다. 모든 패키지들이 데이터베이스에서 SQL 쿼리를 통해 데이터를 선택하고 데이터 프레임이나 그 일부(보통은 행들의 집합으로 구성된)로써 결과를 출력하기 위한 함수를 가지고 있습니다.

RODBC 를 제외한 모든 패키지들이 특정한 DBMS 에 연계되어 있습니다. 그러나 진행중인 작업들은 가장 발달된 형태인 RMySQL 과의 결합에서 시초의 팩키지인 DBI 로 통합하려는 방향이 있습니다(http://developer.r-project.org/db).

또한 CRAN 에는 가장 발달된 형태 중 하나인 ROracle, RPostgreSQL 그리고 RSQLite(번들화 되어 있는 DBMS SQLite 과 같이 작동하는)이 있습니다(http://www.hwaci.com/sw/sqlite).

초기의 두 개 패키지 RmSQL 과 RPgSQL 은 현재 지원되지 않고 CRAN 상의 아카이브 영역에 있습니다. BioConductor 프로젝트는 RdbPgSQL 로 CRAN 상에 업데이트 되어 있습니다. PL/R(http://www.joeconway.com/plr/)은 R 을 PostgreSQL 에 심기위한 프로젝트 입니다.

4.3.1 Packages DBI and RMySQL

CRAN 에 있는 패키지 RMySQL 는 MySQL 데이터 베이스 시스템을 위한 인터페이스를 제공합니다(이에 관하여는 http://www.mysql.com and Dubois, 2000.을 참고하십시오). 여기에서 설명하는 것은 버전 0.5-0 에 관한 것이고 그 이전 버전은 완전히 다른 인터페이스를 가지고 있습니다. 현재 버전은 DBI 패키지를 필요로 하고 여기서의 설명은 다른 모든 최근의 작은 변화들과 함께 DBI 에 적용됩니다.

2001 년 1 월의 3.23.x 버전부터 GPL 하에서 공개된 MySQL 은 Unix/Linux 및 Windows 에 존재합니다. MySQL 은 '가볍고 간결한' 데이터 베이스 입니다(그것은 운용 중인 파일 시스템이 경우에 따라 민감한 이름의 경우를 유지하여 Windows 에서는 작동되지 않습니다). 패키지 RMySQL 은 Linux 와 Windows 모두에서 사용되고 있습니다.

dbDriver("MySQL")의 호출은 데이터베이스 연결 객체를 반환하고 dbConnect 의 호출은 데이터베이스 연결(이것은 범용적 함수인 dbDisconnect 를 호출함으로써 닫는)을 열게 합니다. ROracle, PostgreSQL, 또는 RSQLite 과 같은 각각의 DBMS 와 함께 dbDriver("Oracle"), dbDriver("PostgreSQL") 또는 dbDriver("SQLite")을 사용합니다.

SQL 쿼리들은 dbSendQuery 또는 dbGetQuery 에 의해 보내질 수 있습니다. dbGetQuery 는 쿼리를 보내고 그 결과를 데이터프레임 처럼 추출할 수 있습니다. dbSendQuery 는 쿼리를 내보내고 그 결과들을 추출하는데 종종 사용될 수 있는 "DBIResult"로부터 이어 받은 class 의 객체를 반환하고 그 결과를 삭제하기 위해 dbClearResult 를 호출하는데 사용됩니다.

함수 fetch 는 쿼리의 결과 중 일부분 또는 전체 행을 리스트로 추출하는데 사용됩니다. 모든 행이 fetch 되면 dbHasCompleted 와 dbGetRowCount 는 결과에서 행의 수를 나타내거나 반환합니다.

데이터베이스에서 표를 읽고, 쓰고, 검증하고 그리고 삭제하기 위한 편리한 인터페이스가 있습니다. dbReadTable 과 dbWriteTable 은 MySQL 표에 필드의 행이름을 위해 데이터프레임의 행이름을 하는

```
> library(RMySQL) # will load DBI as well
## open a connection to a MySQL database
> con <- dbConnect(dbDriver("MySQL"), dbname = "test")
## list the tables in the database
> dbListTables(con)
## load a data frame into the database, deleting any existing copy
> data(USArrests)
> dbWriteTable(con, "arrests", USArrests, overwrite = TRUE)
TRUE
> dbListTables(con)
[1] "arrests"
## get the whole table
> dbReadTable(con, "arrests")
Murder Assault UrbanPop Rape
Alabama 13.2 236 58 21.2
Alaska 10.0 263 48 44.5
Arizona 8.1 294 80 31.0
Arkansas 8.8 190 50 19.5
...
## Select from the loaded table
> dbGetQuery(con, paste("select row_names, Murder from arrests",
"where Rape > 30 order by Murder"))
row_names Murder
1 Colorado 7.9
2 Arizona 8.1
3 California 9.0
4 Alaska 10.0
5 New Mexico 11.4
6 Michigan 12.1
7 Nevada 12.2
8 Florida 15.4
> dbRemoveTable(con, "arrests")
> dbDisconnect(con)
```

4.3.2 Package RODBC

CRAN 상의 패키지 RODBC 는 특정한 ODBC 인터페이스를 지원하는 테이터베이스 원천을 위한 인터페이스를 제공합니다. 이것은 매우 광범위하게 적용되고 다른 데이터베이스 시스템에 접속하기 위한 동일한 R 코드를 허용합니다. RODBC 는 Unix/Linux 와 Windows 둘 모두에서 작동하고 거의 모든 시스템이 ODBC 를 위한 지원을 제공합니다.

이제껏 Windows 에서 마이크로소프트의 SQL Server, Access, MySQL 과 PostgreSQL 을 그리고 Linux 에서 MySQL, Oracle, PostgreSQL 과 SQLite 을 시험해 왔습니다. ODBC 는 클라이언트-서버 시스템이어서 Windows 클라이언트에서 Unix 서버에 관해 작동하는 DBMS 에 행복하게 접속해 왔습니다.

Windows 에서 ODBC 지원은 평범하게 설치되었고 현재 버전은 MDAC 의 일부로서 http://www.microsoft.com/data/odbc/에서 사용 가능합니다. Unix 또는 Linux 상에서 여러분은 unixODBC(http://www.unixODBC.org) 또는 iOBDC (http://www.iODBC.org)와 같은 ODBC 드라이버 관리자(driver manager)와 여러분의 데이터베이스 시스템에 적함한 설치된 드라이버가 필요합니다.

Windows 는 DBMS 들을 위한 것뿐만 아니라 엑셀시트('.xls), DBase('.dbf') 그리고 텍스트 파일을 위한 드라이버도 제공합니다(명명된 응용프로그램들이 설치될 필요는 없습니다. 그리고 그런 파일 형태들은 드라이버 버전에 따라 지원 여부가 결정됩니다). 엑셀 2007 과 엑세스 2007 을 위한 '2007 Office System Driver 가 있습니다(http://download.microsoft.com 을 방문하여 'Office ODBC'를 검색하면 'AccessDatabaseEngine.exe'를 찾을 수 있습니다).

동시에 많은 연결(connections)을 하는 것도 가능합니다. 연결은 데이터베이스가 대화상자를 통해 수집된 되도록 하는 Windows GUI 에 따라 odbcConnect 나 odbcDriverConnect 를 호출하여 열고 이후의 데이터베이스 접속을 위해 사용된 것(handle)을 반환합니다.

연결은 close 나 odbcClose 를 호출하거나 또한 (경고 메세지와 함께) R 객체가 아닌 것이 그것을 참조하거나 특정 R 세션의 마지막에 닫습니다.

함수 sqlSave 는 데이터베이스에서 R 데이터프레임을 표로 복사하고 sqlFetch 는 데이터베이스 내의 표를 특정한 R 데이터프레임으로 복사합니다. 어떤 SQL 쿼리는 sqlQuery 를 호출하여 데이터베이스로 보내질 수 있습니다. 이것은 R 데이터프레임에서 결과를 반환합니다(sqlCopy 는 쿼리를 데이터베이스로 보내고 데이터베이스에서 표처럼 결과물을 저장합니다).

보다 좋은 수준의 통제는 최초에 odbcQuery 를 호출함으로써 달성되며 sqlGetResults 는 결과를 불러옵니다. 후자는 함수 sqlFetch 의 기능처럼 반복되는 과정에서 일시에 제한된 수의 행들을 추출하는데 사용될 수 있습니다.

여기에 ODBC 드라이버가 행과 데이터프레임명을 낮은 사례로 포착하는데 PostgreSQL 을 이용하는 사례가 있습니다. 우리는 이전에 생성한 testdb 라는 데이터베이스를 사용하고 DSN(데이터 소스명)을 unixODBC 에서 '~/.odbc.ini'에 준비하였습니다. 정확히 동일한 코드가 Linux 나 Windows 에서 MySQL 데이터베이스에 접속하기 위해 MyODBC 를 사용하여 작동되었습니다(MySQL 은 또한 이름을 낮은 케이스로 파악합니다).

Windows 에서 DSN 들은 관리자(the Control Panel)에서 ODBC 애플릿에 준비됩니다(2000/XP 에서 '응용도구(Administrative Tools)' 섹션에 'Data Sources(ODBC)').

```
> library(RODBC)
## tell it to map names to l/case
> channel <- odbcConnect("testdb", uid="ripley", case="tolower")
## load a data frame into the database
> data(USArrests)
> sqlSave(channel, USArrests, rownames = "state", addPK = TRUE)
> rm(USArrests)
## list the tables in the database
> sqlTables(channel)
TABLE_QUALIFIER TABLE_OWNER TABLE_NAME TABLE_TYPE REMARKS
1 usarrests TABLE
## list it
> sqlFetch(channel, "USArrests", rownames = "state")
murder assault urbanpop rape
Alabama 13.2 236 58 21.2
Alaska 10.0 263 48 44.5
...
## an SQL query, originally on one line
> sqlQuery(channel, "select state, murder from USArrests
where rape > 30 order by murder")
state murder
1 Colorado 7.9
2 Arizona 8.1
3 California 9.0
4 Alaska 10.0
5 New Mexico 11.4
6 Michigan 12.1
7 Nevada 12.2
8 Florida 15.4
## remove the table
> sqlDrop(channel, "USArrests")
## close the connection
> odbcClose(channel)
```

Windows 에서 엑셀작업시트로 ODBC 를 사용하는 단순한 예로 우리는 작업시트에서 다음과 같이 읽을 수 있습니다.

```
> library(RODBC)
> channel <- odbcConnectExcel("bdr.xls")
## list the spreadsheets
> sqlTables(channel)
TABLE_CAT TABLE_SCHEM TABLE_NAME TABLE_TYPE REMARKS
1 C:\\bdr NA Sheet1$ SYSTEM TABLE NA
2 C:\\bdr NA Sheet2$ SYSTEM TABLE NA
3 C:\\bdr NA Sheet3$ SYSTEM TABLE NA
4 C:\\bdr NA Sheet1$Print_Area TABLE NA
## retrieve the contents of sheet 1, by either of
> sh1 <- sqlFetch(channel, "Sheet1")
> sh1 <- sqlQuery(channel, "select * from [Sheet1$]")
```

표를 특정하는 것과 sqlTables 에 의해 반환된 이름은 다르다는 것에 주의하십시오. sqlFetch 는 그 차이를 파악할 수 있습니다.

5. Binary files

Binary connections(Chapter 6 [Connections], Page ?)는 binary file 을 다루는데 현재 가장 선호되는 방법입니다.

5.1 Binary data formats

CRAN 에서 제공되는 **hdf5**, **RNetCDF** 그리고 **ncdf** 패키지는 **NASA 의 HDF5**(Hierarchical Data Format: 위계적 데이터 포맷; 이에 관하여는 http://hdf.ncsa.uiuc.edu/HDF5/를 참고하십시오)와 **UCAR 의 netCDF** 데이터 파일 (network Common Data Form: 네트워크 공용 데이타 폼; 이에 관하여는 http://www.unidata.ucar.edu/packages/netcdf/를 참고하십시오)로의 인터페이스를 제공합니다.

이 둘은 모두 데이터에 대한 개략적 설명(descriptions), 라벨(labels), 포맷(formats), 단위(units) 등을 포함하는 과학적 데이터를 배열기준(array-oriented)의 방법으로 저장하는 시스템입니다. **HDF5** 는 또한 배열(arrays)을 그룹화하고, R interface 는 **HDF5** 그룹에 대한 리스트를 확인(map)하게 할 수 있으며, 숫자나 문자 벡터나 행렬을 사용할 수 있습니다.

CRAN 에서 제공되는 **ncvar** 패키지는 **RNetCDF** 를 통한 **netCDF** 에 대한 high-level R interface 를 제공합니다. **rhd5** 패키지는 http://www.bioconductor.org 에서 구할 수 있습니다.

5.2 dBase files(DBF)

dBases 는 Ashton-Tate 에 의해 만들어진 DOS 프로그램입니다. 그리고 이후에 '**.dbf**'라는 흔히 사용되는 파일 확장자와 함께 flat-file format 의 binary 를 가진 Borland 가 소유하게 되었습니다. 그것은 dBase, Clipper, FoxPro 와 그것들의 윈도우 형태인 Visual dBase, Visual Objects 그리고 Visual FoxPro 를 포함하는 'Xbase'라는 데이터베이스 군에 적용되어 왔습니다(이에 관하여는 http://www.e-bachmann.dk/docs/Xbase.htm 을 참고하십시오). dBase 파일은 헤더(header)와 일련의 필드(fields)를 포함하고 있어서 R 데이터 프레임과 가장 유사합니다. 데이터는 텍스트 포맷으로 저장되며, 문자, 논리적 필드와 수리적 필드, 그리고 최신 버전의 다른 유형들(이에 관하여는 http://clicketyclick.dk/docs/data_types.html 를 참고하십시오)을 포함할 수 있습니다.

read.dbf 와 **write.dbf** 함수(function)는 모든 R platform 에 기본적인 DBF 파일은 읽고 작성할 수 있는 방법을 제공합니다. **RODBC** package 내의 **odbcConnectDbase** 는 윈도우 이용자들에게 마이크로소프트가 제공하는 dBase 드라이버를 통해 DBF 파일을 읽는데 보다 포괄적인 도구들을 제공합니다(또한 Visual FoxPro 드라이버도 **odbcDriverConnect** 를 통해 사용가능 합니다).

6. Connections

R 에서 사용되는 Connections 는 Chambers(1998)가 고안한 것으로, 파일과 같은 것들에 대한 유연한 interface 에 의해서 파일이름의 사용을 바꾸는 함수(function)들의 집합입니다.

6.1 Types of connections

가장 익숙한 유형의 connection 은 파일일 것입니다. 파일 connection 은 **file** 이라는 함수(function)에 의해 생성됩니다. 이러한 file connection 은 사용자의 OS 가 특정 파일의 사용을 지원한다면 text mode 나 binary mode 모두에서 읽기, 쓰기 또는 붙이기를 하는데 사용할 수 있습니다. 실제로 파일들은 읽기와 쓰기 둘 모두를 위해서 사용될 수 있으며, R 은 읽기와 쓰기를 위해 따로 분리된 파일 위치(file position)를 유지합니다.

주의할 점은 default 에서 connection 은 그것이 생성될 때 열리지 않는다는 것입니다. connection 이 사전에 열려있지 않으면 그 connection 을 사용하는 어떤 함수(function)는 그 connection 을 열어야만 하고, 그리고 만약 어떤 함수(function)가 connection 을 열어놓은 상태이면 그 함수(function)의 사용 후에 그 connection 을 닫아야만 하는 것입니다. 간단히 말하면, 당신이 만들어 놓은 상태 그대로 connection 을 유지하십시오. 일반적으로 connection 을 분명하게 열거나 닫는 방법으로 **open** 과 **close** 라는 함수(function)를 사용할 수 있습니다.

gzip 알고리즘을 통해 압축된 파일은 **gzfile** 함수(function)에 의해 생성된 connection 으로 사용될 수 있고, **bzip2** 알고리즘으로 압축된 파일은 **bzfile** 을 통해 사용될 수 있습니다.

Unix 프로그래머들은 종종 **stdin**, **stdout** 그리고 **stderr** 과 같은 특별한 파일들을 다루기도 합니다. 이러한 파일들은 R 에서 **terminal connections** 로 존재합니다. 이것들은 일반적인 파일들이겠지만, **GUI console** 에서 입 • 출력하는 데 사용될 수도 있습니다(the standard R interface 에 의해서도 **stdin** 은 특정 파일이라기 보다는 **readline** 에 의해 line 을 삽입하는데 사용됩니다).

세 개의 terminal connection 은 항상 열려있으며, 따로 열거나 닫을 수 없습니다. **stdout** 와 **stderr** 은 각각 전통적으로 일반적인 출력결과 및 에러 메시지에 각각 사용되어 왔습니다. 이것들은 일반적으로 같은 장소에 저장되지만, 일반적인 출력결과는 **sink** 를 이용하여 다른 디렉터리에 저장할 수 있습니다. 에러 메시지도 **sink, type="message"**의 방법으로 다른 디렉터리를 지정하지 않는 한 **stderr** 에 저장됩니다. 여기서 사용된 말의 의미에 주의하십시오. connection 은 디렉터리를 다른 곳으로 지정할 수 없지만, 출력결과는 다른 connection 에 저장될 수 있습니다.

text connection 은 입력을 할 수 있는 또 다른 방법으로 사용됩니다. text connection 들은 마치 text 파일에서 line 들이 읽히는 것처럼 R 문자 벡터가 읽히도록 해 줍니다. text connection 은 **textConnection** 을 호출(call)해서 생성되고 열립니다. 그리고 text connection 은 생성과 동시에 문자 벡터의 현재 내용을 내부버퍼(internal buffer)에 복사합니다.

text connection 은 R 의 출력결과를 문자 벡터로 인식하는 데에도 사용될 수 있습니다. **textConnection** 은 사용자의 작업공간(user's workspace)에서 새로운 문자 object 를 생성하거나 현재 존재하고 있는 것에 추가하는 두 가지 경우 모두에 사용될 수 있습니다. 이 connection 은 **textConnection** 의 호출(call)에 의하여 열리고, connection 에 있는 완전한 행 출력결과(complete lines output)는 항상 R object 에서 사용가능 합니다. 이 connection 을 닫는 것은 남아있는 어떤 출력결과를 문자벡터의 마지막 요소로 기록합니다.

Pipes 는 다른 과정(another process)과 연결시켜주는 특별한 형태의 파일 중의 하나이며, pipe connection 은 **pipe** 함수에 의해 생성됩니다. 쓰기(writing)를 위한 pipe connection 의 개시(opening)는 어떤 OS 명령을 실행(pipe 에 추가하는 것은 아닙니다)하고, 그것의 표준적 입력을 모든 R 이라도 그 connection 에 쓰는 것과 연결시켜 줍니다. 반대로 말하면, 입력을 위한 pipe connection 의 개시는 어떤 OS 명령을 실행하고 그것의 표준 출력결과를 그 connection 으로부터 R 의 입력으로 사용할 수 있게 합니다.

'**http://**', '**ftp://**', 그리고 '**file://**'과 같은 유형의 URL 들은 **url** 함수를 사용함으로써 읽혀질 수 있습니다. 편의를 위해 파일을 지정함으로써 **file** 또한 '**http://**', '**ftp://**', 그리고 '**file://**'을 인식하고 **url** 을 불러옵니다.

Socket 은 Berkeley 와 같은 socket 을 지원하는 platform(대부분 Unix, Linux 그리고 Windows)에서 **socketConnection** 함수를 통해 connection 들처럼 사용될 수 있습니다. Socket 은

client 와 server socket 들이 사용될 수 있으며 이들 client 와 server socket 에 쓰거나 이들로부터 읽어들일 수 있습니다.

6.2 Output to connections

지금까지 파일에 쓰기(writing)의 방법으로 함수 **cat**, **write**, **write.table** 그리고 **sink**, 파일에 붙여넣기(appending)의 방법으로 인수(argument) **append=TRUE** 에 대하여 설명하였습니다. 이것은 R 1.2.0 이전 버전에서 작동합니다.

현재의 방법도 동일하지만, 실제로 일어나는 것은 **file** 인수(argument)가 문자열(a character string)일 때, 파일 connection 이 열리고 함수의 호출이 끝날 때 다시 닫힙니다. 만약 우리가 반복적으로 같은 파일을 쓰고 싶다면, 그 connection 을 명백히 선언(declare)하여 열고 각 output 함수를 호출하는 connection object 를 건너뛰는 방법이 효율적입니다. 이것은 또 pipe 에 쓰는 것을 가능하게 합니다. 지금도 여전히 가능하지만 이것은 전에는 syntax **file="| cmd"**를 이용한 제한된 방법으로만 사용할 수 있었습니다.

어떤 connection 에 완전한 텍스트 line 을 쓰는 **writeLinesfk** 라는 함수가 있습니다.
간단한 예는 아래와 같습니다.

```
zz <- file("ex.data", "w") # open an output file connection
cat("TITLE extra line", "2 3 5 7", "", "11 13 17",
file = zz, sep = "\n")
cat("One more line\n", file = zz)
close(zz)
## convert decimal point to comma in output, using a pipe (Unix)
## both R strings and (probably) the shell need \ doubled
zz <- pipe(paste("sed s/\\\\./,/ >", "outfile"), "w")
cat(format(round(rnorm(100), 4)), sep = "\n", file = zz)
close(zz)
## now look at the output file:
file.show("outfile", delete.file = TRUE)
## capture R output: use examples from help(lm)
zz <- textConnection("ex.lm.out", "w")
sink(zz)
example(lm, prompt.echo = "> ")
sink()
close(zz)
## now 'ex.lm.out' contains the output for futher processing.
## Look at it by, e.g.,
cat(ex.lm.out, sep = "\n")
```

6.3 Input from connections

connection 으로부터 읽는 기본적 함수는 **scan** 과 **readLines** 입니다. 이것들은 문자열(character string) 인수(argument)를 취하고 함수의 지속을 위해 파일 connection 을 엽니다. 그러나 분명한 파일 connection 의 열기(opening)는 다른 format 에서 연속적으로 파일 읽기를 가능하게 해줍니다.

scan 을 호출하는 다른 함수들 또한 connection 을 사용할 수 있습니다. 특히 **read.table** 이 그렇습니다. 간단한 예는 아래와 같습니다.

```
## read in file created in last examples
readLines("ex.data")
unlink("ex.data")
## read listing of current directory (Unix)
readLines(pipe("ls -1"))
# remove trailing commas from an input file.
# Suppose we are given a file 'data' containing
450, 390, 467, 654, 30, 542, 334, 432, 421,
357, 497, 493, 550, 549, 467, 575, 578, 342,
446, 547, 534, 495, 979, 479
# Then read this by
scan(pipe("sed -e s/,$// data"), sep=",")
```

편의를 위해, 만약 **file** 인수(argument)가 어떤 FTP 나 HTTP URL 을 특정한다면, 그 URL 은 읽기(reading)를 위해 **url** 을 통해 열립니다. '**file://foo.bar**'를 통해 파일을 특정하는 것도 허용됩니다.

6.3.1 Pushback

C 프로그래머들은 텍스트 입력 stream 에 어떤 문자를 뒤로 밀리게 하는 **ungetc** 에 익숙할 것입니다. R connection 은 더 강력한 방법으로 같은 작업을 할 수 있는데, **pushBack** 의 호출을 통하여 connection 에 임의의 수만큼 텍스트 line 을 뒤로 밀리게 할 수 있습니다.

Pushback 은 stack 처럼 동작해서, 첫 번째 읽기 요구는 가장 최근에 pushback 된 text 에서 각 line 을 이용하고, 그 다음엔 이전의 pushback 된 text, 마지막으로 connection 자체로부터 읽어 들입니다. 한 번 pushback 된 line 은 완벽히 읽어나서 지워집니다. push back 되고 있는 line 의 수는 **pushBackLength** 을 호출하여 알 수 있습니다. 간단한 예는 아래와 같습니다.

```
> zz <- textConnection(LETTERS)
> readLines(zz, 2)
[1] "A" "B"
```

```
> scan(zz, "", 4)
Read 4 items
[1] "C" "D" "E" "F"
> pushBack(c("aa", "bb"), zz)
> scan(zz, "", 4)
Read 4 items
[1] "aa" "bb" "G" "H"
> close(zz)
```

Pushback 은 텍스트 모드에서 입력 할 때 connection 이 열려있을 경우에만 사용가능 합니다.

6.4 Listing and manipulating connections

User 에 의해 현재 열려있는 모든 connection 의 개요는 **showConnections( )**로 알아낼 수 있고, 닫혀 있는 connection 과 terminal connection 등을 포함한 모든 connection 의 개요는 **showConnections(all=TRUE)**로 알 수 있습니다.

일반적 함수인 **seek** 은 읽기에 사용될 수 있으며 (어떤 connection 에서는) 읽기 또는 쓰기와 같은 현재의 position 을 reset 할 수 있습니다. 그런데 이것은 불행하게도 신뢰하기 어려울 수 있는 OS facility 에 따라 달라집니다(예를 들어, Windows 환경에서의 text 파일). 함수 **isSeekable** 은 함수 **seek** 이 position 바꿀 수 있는 지를 보고해 줍니다.

함수 **truncate** 는 현재의 position 에서 읽기나 쓰기를 위해 열려있는 파일을 truncate 하는데 사용될 수 있습니다. 이것은 오로지 **file** connection 에 대해서만 작동하고, 역시 모든 platform 에서 적용되지는 않습니다.

6.5 Binary connections

readBin 과 **writeBin** 은 binary connection 에서 읽거나 쓰는데 사용됩니다. binary 모드에서 "**b**"를 모드 specification 에 덧붙임으로써 connection 이 열립니다. 즉 모드 "**rb**"는 읽기에 그리고 모드 "**wb**" 또는 "**ab**"는 쓰기에 사용합니다. 이 함수들은 다음과 같은 인수(argument)를 가집니다.

```
readBin(con, what, n = 1, size = NA, endian = .Platform$endian)
writeBin(object, con, size = NA, endian = .Platform$endian)
```

각각의 경우에 **con** 은 호출이 유지되는데 필요한 경우 열려 있으며 문자열이 주어지면, 파일명을 지정하는 것으로 가정됩니다.

쓰기(writing)를 묘사하는 것은 조금 더 단순하여서 우리는 먼저 그것을 설명할 것입니다. **object** 는 atomic vector object 로 attributes 가 없는 **numeric**, **integer**, **character**, **complex** 또는 **raw** 와

같은 mode 의 vector 입니다. 디폴트 상태에서 이것은 memory 에 나타나는 것과 똑같이 바이트의 흐름처럼 파일에 쓰여집니다.

readBin 은 파일로부터 바이트의 흐름을 읽고 그것을 **what** 으로 주어진 mode 의 vector 처럼 해석합니다. 이것은 적절한 mode 의 object(예를 들어, **what=integer( )**)나 또는 그 mode 를 묘사하는 문자열(character string)이 될 수 있습니다(앞의 문단에서 말한 다섯 가지 중 하나 또는 **"double"** 또는 **"int"**). 인수(argument) n 은 connection 으로 부터 읽기 위기 위한 벡터 요소의 최대 수를 지정합니다. 즉, 보다 적은 수가 가능하면 더 짧은 벡터가 만들어 집니다. 인수(argument) **signed** 는 1 바이트와 2 바이트의 정수가 default 인 기호화된 정수 또는 기호화되지 않은 정수처럼 읽혀지게 합니다.

남아있는 두 인수(argument)는 데이터를 다른 프로그램이나 다른 플랫폼으로 대체시켜 쓰거나 읽는데 사용됩니다. 디폴트 상태의 바이너리 데이터(binary data)는 직접 메모리에서 connection 으로 또는 그 반대로 옮겨 집니다. 이것은 특정 파일이 다른 구조로 특정 장치(machine)로 옮겨져야 한다면 충분치 않을 것이나 거의 모든 R 플랫폼들 사이에서는 단지 바이트 순서(byte-order)만 변화시켜주면 됩니다. 일반적인 PC('ix86'과 'x86_64' 기반)들과 Compaq Alpha 그리고 Vaxen 은 little-endian 구조인데 반해, Sun Sparc, mc680x0 시리즈, IBM R6000, Apple Macintosh, SGI 등의 다른 대부분의 장치들은 big-endian 구조 입니다(네트워크 바이트 순서(network byte-order)는 XDR(eXternal Data Representation)에서 사용되는 것과 같이 big-endian 입니다.) 예를 들어 16 비트로 된 정수나 단정밀도(single-precision; 단정도 또는 단밀도) 실수를 쓰는 경우와 같이 다른 프로그램으로 전송하거나 받아오기 위해서 우리는 조금 더 작업을 할 필요가 있을 지도 모릅니다. 이것은 **size** 인수(argument)를 이용함으로써 해결할 수 있는데, **size** 인수(argument)는 정수와 논리자(logical)의 크기를 1, 2, 4, 8 또는 4, 8 그리고 또는 실수를 12 나 16 으로 정할 수 있게 합니다. 다른 크기로 전환하는 것은 정확성을 잃을 수 있으므로 결측값(NA: not available)들을 포함하고 있는 벡터에 대해서는 사용하지 말아야 합니다.

문자열은 C 포맷에서 읽히거나 쓰여집니다. 그것은 바이트 열(string of bytes)처럼 영(0) 바이트에 의해 종료됩니다. 함수 **readChar** 와 **writeChar** 는 보다 더 유연하게 사용할 수 있습니다(Character strings are read and written in C format, that is as a string of bytes terminated by a zero byte. Functions readChar and writeChar provide greater flexibility.)

6.5.1 Special values

크기 변화가 관련되어 있다면 시도되지 말아야 하더라도 함수 **readBin** 과 **writeBin** 은 결측값(missing values)과 특이값(special values)을 처리해 줍니다.

R 에서 논리 유형(logical type)과 정수 유형(integer type)의 경우 결측값(missing values)은 '**limits.h**'로 C 의 헤더로 정의된 나타낼 수 있는 가장 작은 **int** 인 **INT_MIN** 인데 이것은 보통 비트 패턴(the bit pattern) 0x80000000 에 대응하는 것 입니다.

R 에서 숫자와 복합적인 형태의 특이값(special values)의 표현은 사용하는 컴퓨터(machine)나 컴파일러(Compiler: BASIC, COBOL, PASCAL 등의 프로그래밍 언어를 기계어로 번역하는 프로그램)에 따라 달라집니다(The representation of the special values for R numeric and complex types is machine-dependent, and possibly also compiler-dependent.) 그것들을 이용하는 가장 단순한 방법은 이중상수(double constants)인 **NA_REAL**, **R_PosInf** 그리고 **R_NegInf** 를 내보내고 **ISNAN** 과 **R_FINITE** 과 같은 매크로를 정의하는 '**Rmath.h**' 헤더를 포함하는 독립적인 **Rmath** 라이브러리 외부의 어플리케이션(application; 응용프로그램?)(an external application)과 연결시키는 것 입니다.

만약 이와 같은 작업이 불가능하면 산술(arithmetic)이 사용되는 모든 보통의 IEC 60559 (aka IEEE 754) 플랫폼에서, Inf, -Inf 그리고 NaN 값들을 테스트하거나 설정하는데 표준적인 C 도구(facility)를 사용할 수 있습니다. 그러한 플랫폼에서 NA 는 low-word 0x7a2(1954 in decimal)에따라 NaN 값으로 표시됩니다.

문자결측값들은 NA 처럼 사용되며, 결측값으로 문자 값을 인식할 수 있는 규칙(provision)은 없습니다(이것은 마치 한 번 읽고 다시 그것들을 재할당함으로써 완성되는 것과 같습니다).

7. Network interfaces

몇 가지 제한된 도구들이 network connection 을 통해 낮은 수준의 데이터를 교환하는데 사용될 수 있습니다.

7.1 Reading from sockets

기본적인 R 프로그램은 그것들을 지원(보통의 Linux, Unix, 그리고 R 의 Windows 포트(port)를 포함)하는 시스템에서 BSD 소켓(sockets)을 통한 커뮤니케이션을 위한 몇 가지 장치(facility)에 연결되어 있습니다. 이러한 소켓들(sockets)을 사용하면서 잠재적으로 발생할 수 있는 문제점 하나는 이러한 장치들이 종종 보안상의 문제로 차단되거나 Web caches 의 사용이 강요된다는 것입니다. 그래서 이러한 함수들은 외면상(externally) 보다는 인터넷에서 더욱 유용할 수 있습니다.

초기 낮은 수준의 인터페이스(interface)는 함수 **make.socket**, **read.socket**, **write.socket**, and **close.socket** 으로 주어졌습니다.

7.2 Using download.file

함수 **download.file** 은 FTP 나 HTTP 를 통한 Web 기반 자원(resource)에서 파일을 읽거나 파일에 쓰기를 위해 제공됩니다. 종종 이것은 read.table 과 같은 함수들과 같이 피할 수 있고, scan 은 명백히 connection 을 열기 위해서 url 을 사용하거나 file 인수(argument)로서 URL 을 부여함으로써 특정 URL 로부터 직접 읽기를 할 수 있게 합니다.

7.3 DCOM interface

DCOM 은 서로 다른 프로그램들-또는 장치들(machines)-간의 커뮤니케이팅(communicatin g)을 위한 Windows 프로토콜입니다. CRAN 의 Software->Other->Non-standard 목록에 있는 Thomas Baier 의 **StatConnector** 프로그램은 패키지 **rscproxy** 에 있는 프록시 DLL 에 대한 인터페이스를 제공하고 DCOM 서버를 만듭니다. 이것은 vector 와 matrix 와 같은 단순한 대상을 통과하고 R 로부터 그리고 R 로 명령을 보내는데 사용될 수 있습니다(이것은 아직 작용하는지는 분명치 않습니다. 다른 버전의 RExcelInstaller 패키지가 있습니다).

이 프로그램은 by Erich Neuwirth available 비쥬얼 베이직 증명(demonstration)과 Excel 플러그인(plug-in)을 수반합니다. 이 인터페이스는 여기에서 다룬 인터페이스의 대부분에 대해 다른 어플리케이션(application; 응용프로그램?)들은 클라이언트로서 그리고 R 은 서버로서 다른 방향(direction)에 있습니다.

다른 (D)/COM 서버는 **http://www.omgahat.org** 애 있고 그것은 R 대상(objects)을 COM 값으로 내보낼 수 있도록 해줍니다. 그 사이트는 또한 패키지 RDCOMClient 와 SWinTypeLibs 를 가지고 있는데 그것은 R 이 (D)COM 처럼 작동하게 해줍니다.

7.4 CORBA interface

CORBA(Common Object Request Broker Architecture)는 어플리케이션(application; 응용프로그램?)이 잠재적으로 다른 언어와 다른 장치에서 작동중인 프로그램되어 다른 어플리케이션(application; 응용프로그램?)들에서 작동중인 서버 대상(server objects)에서 방법 또는 오퍼레이션들(?)을 불러들일 수 있게 한다는 점에서 DCOM 과 비슷합니다. 패키지 CORBA 는 Omegahat project(http://www.omegahat.org/RSCOR BA/)에서 얻을 수 있으며, 현재 Unix 버전이 제공되고 있습니다. Windows 버전은 제공되고 있지 않지만 앞으로 개발할 것으로 보입니다.

이 패키지는 R 명령어들이 사용 가능한 CORBA 서버에 접속하고, 그것들이 제공하는 방법들(methods)에 대해 질문(query)하고, 그리고 동적으로(dynamically) 이런 대상들에 대한 방법을 불러 들입니다(invoke). 이러한 불러오기(calls)에서 인수로 주어진 R 값들은 불러오기(the call)에서 내보내어지고 오퍼레이션(?) 불러오기(invocation: 조회?)를 가능하게 만들어 줍니다. 원시 데이터 유형(vector 와 list)은 디폴트 형태로 내보내기 됩니다. 반면, 더욱 복잡한 형태의 대상은 참조(reference)에 따라 내보내기 됩니다. 이것의 사용사례는 Gnumeric(http://www.gnumeric.org) 스프레드시트(spreadsheet)와의 커뮤니케이팅과 데이터 시각화 시스템 ggobi 와의 상호작용을 포함합니다.

또한 누구든지 다른 어플리케이션들이 이런 방법을 요구하도록 함으로써 R 에서 CORBA 서버를 생성할 수 있습니다. 예를 들어, 누군가 특정 데이터세트(dataset) 또는 어떤 R 모델링 소프트웨어 접근을 신청할 수 있습니다. 이것은 R 데이터 대상과 함수를 결합함으로써 동적으로 이루어집니다. 이것은 누구든지 명백하게 데이터와 R 로부터의 기능성(functionality from R)을 내보낼 수 있게 합니다.

누구든지 R 에서 computing 하는 것과 동일(parallel)하게 배포할 수 있도록 CORBA 패키지를 이용할 수 있습니다. 어느 R 세션은 마치 매니저처럼 행동하기도 하고 다른 R 작업자(worker) 세션에서 작동하는 다른 서버에 대해 일을 처리할 수 있습니다. 이것은 R 에서 비동기(asynchronous) 또는 background CORBA call 을 부르는 능력을 사용합니다. 이것에 대한 더 자세한 정보는 Omegahat project(http://www.omegahat.org/RSCORBA)에서 얻을 수 있습니다.

8. Reading Excel spreadsheets

가장 일반적인 R 데이터 불러오기/내보내기 질문은 '어떻게 엑셀에서 데이터를 가져오지?'하는 것입니다. 이 장은 도움마라과 앞서 설명한 옵션을 함께 모아둔 것입니다. 도움말의 대부분이 엑셀 2007 이전의 스프레드시트를 위한 것임에 유의하십시오. 현재 '.xlsx' 포맷의 파일을 읽는 유일한 방법은 RODBC 를 통하는 것입니다.