R 기반의 데이터 시각화

전희원

http://freesearch.pe.kr

사랑하는 보현이와 수빈, 우빈에게 ...

저작권 고지

이 문서의 저작권은 저자에게 있으며 원작을 훼손하거나 개작을 하는 것을 금지합니다. 최신의 pdf파일은 `http://freesearch.pe.kr`에서 무료로 제공됩니다.

Contents

1 R로 하는 데이터 시각화의 시작 **2**
1.1 시각화의 중요성 2
1.2 몇가지 시각화의 예 3
1.3 앞으로 방향 6

2 R 프로그래밍 **9**
2.1 인터렉티브 분석(interactive analysis)과 R 10
2.2 R을 설치하자 11
2.3 R GUI와 IDE환경 소개 12
2.4 RStudio를 이용하자 14
2.5 R 프로그래밍 15

3 Data munging with R **39**
3.1 들어가며 39
3.2 R BASE 집계 함수 소개 40
3.3 tapply, aggregate, by 함수 45
3.4 plyr 패키지 49
3.5 data.table 패키지 52
3.6 왜 멍잉(munging)을 하느냐? 55

4 ggplot2를 이용한 R 시각화 **57**
4.1 왜 ggplot2이 필요하나? 57
4.2 문법(GRAMMAR OF GRAPHICS) 61
4.2.1 레이어를 이용한 ggplot2 시각화 65
4.2.2 GEOM 67
4.2.3 STAT 68

4.2.4 위치 조정 . . . 70
4.2.5 GEOM과 STAT의 결합 . . . 70
4.3 마지막 예제 . . . 73
4.4 장을 마치며 . . . 75

5 잉크스케이프를 활용한 그래프 후처리 77
5.1 환경 설정하기 . . . 77
5.2 예제 그래프 만들기 . . . 78
5.3 잉크스케이프로 그래프 후처리하기 . . . 80
5.4 그래프 후처리와 나머지 작업 . . . 83

6 R로 그래프 플로팅을 하기 위한 몇 가지 팁 85
6.1 웹으로 게시할 그래프에 J(E)PG를 사용하지 말자. . . . 85
6.2 anti-aliasing을 활성화 하라. . . . 87
6.3 정확한 디바이스 드라이버를 사용해 그래프를 저장하라. . . . 88
6.4 필요시, 고해상도 이미지로 출력하라. . . . 88
6.5 출력을 위해서라면 PDF를 활용하라 . . . 89

Abstract

이 문서는 R(R Core Team, 2012)을 사용해 시각화를 하는데 필요한 제반 사항들을 설명한다. 시각화가 왜 필요한지에 대한 소개와 더불어 시각화를 위한 데이터 전처리를 설명하고 시각화의 핵심 부분은 ggplot2(Wickham, 2009)을 활용하게 된다. 이후 그래프 후처리에 대한 내용을 Inkscape[1]를 기반으로 설명하게 되며, 마지막은 미려한 그래프를 만들기 위한 몇가지 팁에 대한 설명을 하도록 하겠다.

[1] http://inkscape.org/

Chapter 1

R로 하는 데이터 시각화의 시작

1.1 시각화의 중요성

최근 빅 데이터 붐이 일어나면서 데이터 처리 플랫폼인 Hadoop[1]과 함께 R(R Core Team, 2012)이라는 언어도 뜨고 있으며 빅 데이터의 분석 방법으로 다시 각광을 받게 되는 게 데이터 시각화(data visualization)이다. 왜 빅 데이터에서 데이터 시각화가 각광을 받는지 잘 생각해 보면 이렇다.

데이터가 굉장히 커서 이들 개개를 일일이 살펴보는 건 거의 불가능에 가까운데, 그렇다고 단순히 평균, 표준편차 등의 요약된 통계량을 가지고 보는 것도 사실 분석가에게는 어느 정도 편리한 요약 수단이라고 해도, 이를 기반으로 누군가를 설득하기 위해서는 사람들마다 공통적으로 가지고 있는 훌륭한 인지 기관인 눈에 호소할 수 있는 데이터 시각화가 분석과 타인을 설득하기 위해 가장 효과적인 방법 중에 하나가 된다. 아무리 데이터를 빨리 처리하는 Hadoop플랫폼이 있다고 하더라도 결과적으로 분석한 것을 정보로 만들지 않으면 그저 거대한 데이터로 밖에 남아 있지 않을 것이다.

필자가 빅 데이터 플랫폼을 모 금융 회사에서 금융 데이터를 기반으로 파일럿을 진행해본 경험을 이야기 해보자면 빅 데이터 처리 플랫폼이 아무리 빨리 처리를 하더라도 처리한 결과의 정보가 의미가 있어야 빅 데이터 플랫폼의 가치를 사람들이 인지한다는 것을 알 수 있었는데, 그 정점에 서 있는 기술이 바로 시각화 기술이었다. "많은 데이터를 처리한 결과, 우리가 얻는 정보나 혜택이 뭐냐?" 하고 물었을 때 바로 답변을 해줄 수 있는 몇 가지 안 되는 기술 중에 하나가 바로 시각화였고, 앞으로의 강좌에서 설명할 ggplot2(Wickham, 2009)를 활용해서 잘 마무리한 경험이 있었다. 한마디로 빅 데이터 플랫폼 엔지니어 뿐만 아니라 데이터를 다루는 사람들은 배워볼 필요가 충분한 기술이라는 것이다.

이제 본격적으로 시각화에 대한 맛을 보도록 해보자!

[1] http://hadoop.apache.org/

1.2 몇가지 시각화의 예

한 예로 각 나라의 자살률을 살펴본다고 한다.
데이터는 http://en.wikipedia.org/wiki/List_of_countries_by_suicide_rate에서 가져왔다. 위키에서 보면 데이터의 신빙성에 대한 논란은 있으니 참고 바란다.
이 예제의 제시 목적은 시각화 학습을 위한 목적도 있으나, 한국의 자살률이 세계적으로 높다는 측면을 강조해 경각심을 키우기 위함도 있다. "다른 나라에 비해 우리나라의 자살률이 왜 높을까?" 라는 의문을 독자들이 가지는 순간 아마도 이와 상관된 데이터를 찾아볼 수 있을 것이고 관련 있는 연구의 촉매가 될 수도 있을 거란 소망을 가져본다. 물론 그런 연구에서 시각화가 좋은 도구가 될 것임은 자명할 것이다.
이 데이터에 대한 시각화에 부담을 가지고 있거나 시도를 해볼 여력이 없는 분들은 그저 엑셀에서 10만명당 몇 명이 자살을 하는지에 대해서 내림차순으로 정렬해서 가장 높은 나라들이 어떤 나라들인지 살펴볼 것이다.

	A	B	C	D	E	F	G
1	Rank	Country	Male	Female	Total	Year	
2	1	Lithuania	61.3	10.4	34.1	2009	
3	2	South Ko	41.4	21	31.2	2010	
4	3	Guyana	39	13.4	26.4	2006	
5	4	Kazakhst	43	9.4	25.6	2008	
6	5	Belarus			25.3	2010	
7	6	Hungary	40	10.6	24.6	2009	
8	7	Japan	33.5	14.6	23.8	2011	
9	8	Latvia	40	8.2	22.9	2009	
10	9	People's Republic of China			22.23	2010	
11	10	Slovenia	34.6	9.4	21.9	2009	

그림. 1.1: 엑셀로 그리드로 본 자살율 데이터

물론 아래와 같이 단순한 통계량을 가지고 전체적인 관점으로 여자보다 남자의 자살이 더 많다는 것을 알 수 있으나 어떤 나라에서 그런 차이를 보이는지는 알 수 없다.

```
##       Rank                      Country        Male          Female          Total            Year     
##  Min.   :  1.0   Albania            :  1   Min.   : 0.00   Min.   : 0.00   Min.   : 0.00   Min.   :1978  
##  1st Qu.: 27.5   Antigua and Barbuda:  1   1st Qu.: 5.45   1st Qu.: 1.70   1st Qu.: 3.60   1st Qu.:2006  
##  Median : 54.0   Argentina          :  1   Median :12.30   Median : 3.45   Median : 7.90   Median :2008  
##  Mean   : 54.0   Armenia            :  1   Mean   :14.39   Mean   : 4.14   Mean   : 9.62   Mean   :2006  
##  3rd Qu.: 80.5   Australia          :  1   3rd Qu.:20.18   3rd Qu.: 6.05   3rd Qu.:14.20   3rd Qu.:2009  
##  Max.   :107.0   Austria            :  1   Max.   :61.30   Max.   :21.00   Max.   :34.10   Max.   :2011  
##                  (Other)            :101   NA's   :5       NA's   :5                                      
```

위의 결과에서 얻을 수 있는 건 남, 녀의 자살률의 차이일 뿐인데, 이를 단순히 상자그림 (boxplot[2]) 으로 표현하면 더 보기 쉽다.

[2] http://en.wikipedia.org/wiki/Box_plot

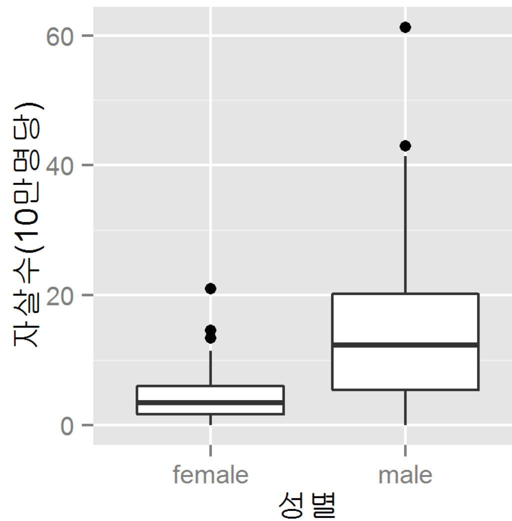

그림. 1.2: 자살율 데이터에 대한 boxplot 결과

그림1.2처럼 시각적으로 데이터를 표현함으로써 많은 정보를 한눈에 볼 수 있게끔 해주는 게 바로 데이터 시각화라는 것이다.
하지만 뭔가 아쉬운 점이 있는데, 실제 아웃라이어(outlier[3]) 라고 불리는 평균이나 중앙값에서 멀리 떨어진 값이 어떤 것인지 표시해 주면 훨씬 가독성이 있을 거라 생각해 본다.

그림1.3와 같이 표현하면 아웃라이어에 어떤 나라들이 있는지 확인이 가능할 것이다. 안타깝지만 여자든 남자든 한국의 자살률이 세계적이라는 것을 볼 수 있는데, 기분 좋지 않은 상위권이 아닐 수 없다. 위 그래프 만으로 데이터 파일의 거의 모든 정보를 보여주는데, 텍스트 크기와 색깔은 10만명당 자살수(남+여) 의 상대적인 크기를 알려주고 있고, 남, 녀 개별적인 자살수도 역시 보여주고 있어 상대적인 비교를 가능하게 도식화 하고 있다.
다른 예를 들자면, 얼마 전에 필자가 블로그에 올려둔 타이타닉 데이터 분석 포스팅을 더 자세히 살펴보자(`http://freesearch.pe.kr/archives/2855`). 이 포스팅의 목적은 타이타닉 사망자 통계를 기반으로 남자의 경우 죽을 확률이 높았고, 여자는 살 확률이 높았다는 것을 데이터를 통해 보여주고, 영화가 비극적인 결말로 끝을 맺을 가능성이 애초부터 높았다는 것을 알려주기 위함이다. 여기서 성별에 따른 생존율이 어떻게 다르고, 승객 등급에 따라 생존율 변화가 어떻게 되는지 가장 먼저 확인하고 싶어질 것이다. 일단 개별적으로 생존율을 확인 하기 위한 아래와 같은 결과를 확인 가능할 것이다.

[3] http://en.wikipedia.org/wiki/Outlier

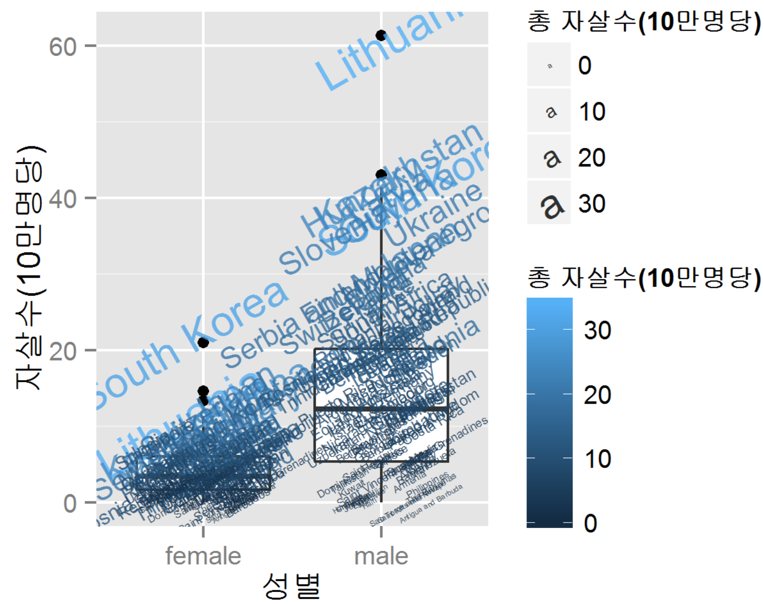

그림. 1.3: 좀더 자세한 자살율 데이터에 대한 boxplot 결과

```
> titanic.dt[, list(prob_as_class=length(which(survived == 1))/nrow(.SD)), by=pclass]

##    pclass prob_as_class
## 1:    1st        0.6192
## 2:    2nd        0.4296
## 3:    3rd        0.2553

> titanic.dt[, list(prob_as_sex=length(which(survived == 1))/nrow(.SD)), by=sex]

##       sex prob_as_sex
## 1: female      0.7275
## 2:   male      0.1910

> titanic.dt[, list(prob_as_sex=length(which(survived == 1))/nrow(.SD)), by=isminor]

##    isminor prob_as_sex
## 1:   adult      0.3658
## 2:   child      0.5596
```

각 등급에 따른 생존율은 위와 같이 숫자로 표현 가능하다(상세한 코드 설명은 1회의 범위에 넘어가니 하지 않겠다). 이를 보면 승객등급(pclass)이 높을수록 생존율이 높다는 것을 알 수 있고, 여자의 경우 남자보다 생존율이 높다는 것과 성인(isminor=adult) 생존율이 높았다는 것을 알 수 있다. 하지만 이들간의 상호작용은 위 통계만으로는 알기 힘들다. 상호작용이라 하면 승객 등급이 여자와 남자의 생존율 차이에 어떤 영향을 줄 수 있느냐이다. 이를 보기 위해서는

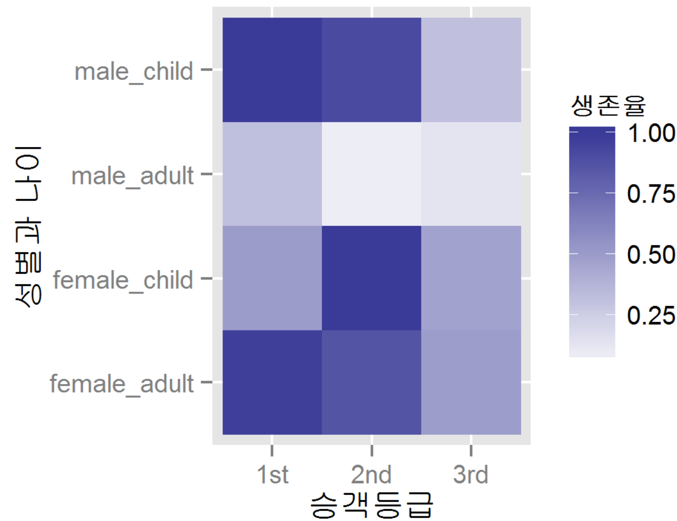

그림. 1.4: 승객등급, 성별나이에 따른 생존율

'chi-square'와 같은 통계적 방법이 필요하나 일반인들에게 이해시키기 위해 부연설명이 너무 길어지기 때문에 필자의 경우에는 시각화를 이용한다.

그림1.4를 보면 Y축에는 성별에 따른 나이를 표기하고, X축은 승객 등급을 디스플레이 했고 범례로 오른쪽 끝에 생존율 범례를 확률값으로 넣었다. 그리고 그래프 내부에서는 박스 형태로 구간을 나눠서 확률 범례를 기준으로 색상의 밝기로 생존율을 표기 했다. 이렇게 되면서 등급과 성별 그리고 성인 유무에 따른 생존율을 한눈에 볼 수 있게 되었다. 이를 볼 때 등급이 올라갈수록 생존율이 높아지는 경향이 있으나 이 또한 성별 그리고 성인유무에 따른 영향을 받는 것으로 볼 수 있다.

그림1.5 그래프는 결정나무(Decision Tree)의 한 알고리즘을 통해 데이터를 분석, 생존에 영향을 미치는 인자들이 어떤 것들이 있는지 나무 모형으로 도식화한 그림이다. 이를 보면 일단 성별이 생존에 가장 큰 영향을 끼치며 이후로는 각 성별마다 다른데 남자는 성인유무, 여자의 경우 승객등급의 영향이 생존에 가장 큰 영향을 끼치는 것으로 볼 수 있다. 게다가 트리 그래프가 타일그림(tile plot)보다는 더 많은 정보를 보여주고 있음을 알 수 있다.

1.3 앞으로 방향

시각화를 하는 방법은 굉장히 많다. 예를 들어 `flowingdata.com`에서 시각화에 어떤 툴을 사용하는지 설문을 한 적이 있었는데, 아래와 그림1.6와 같은 결과를 보여줬다.

Excel과 R이 과반을 넘고 있는 상황을 알 수 있는데, flowingdata 사이트가 시각화에 대한 유

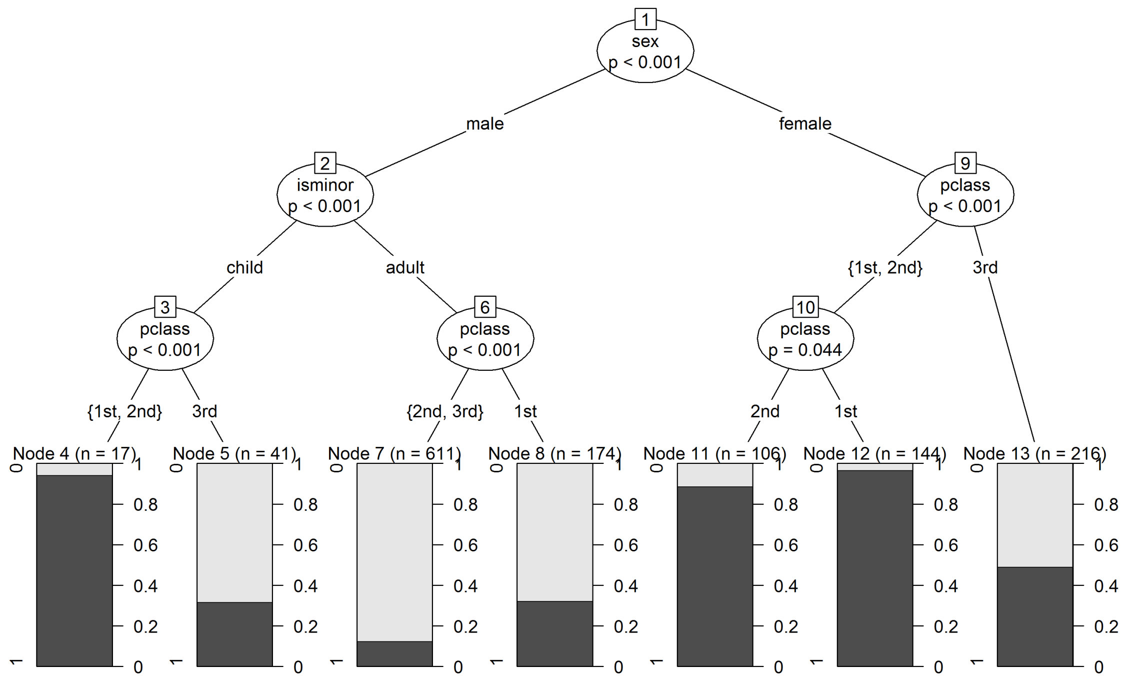

그림. 1.5: Tree 알고리즘을 이용한 생존율에 영향을 미치는 요인 시각화

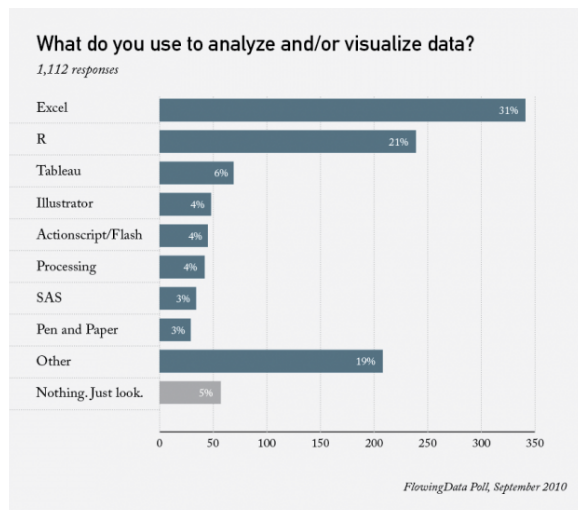

그림. 1.6: flowingdata의 설문 결과

명한 사이트라는 것을 감안할 때 유의미한 설문결과[4]라는 것을 알 수 있다. 사실 Excel은 훌륭한 도구이지만 Excel에 대한 책은 주위에 굉장히 많기도 하여 필자의 시리즈로는 R로 시각화를 하는 방법에 대한 강좌를 열도록 하겠다(게다가 Excel은 다룰 수 있는 데이터 레코드 개수에 한계가 있다는 단점이 존재한다). 데이터를 시각화 하기 위한 세 단계는 아래와 같다.

1. 데이터를 (어떻게든지) 구한다.
2. 데이터를 시각화 하기 위한 데이터로 변형시킨다.
3. 데이터를 시각화 한다.
4. 시각화 꾸미기(옵션)

위의 과정을 위해서는 다양한 지식이 필요하다. 그러나 다행스럽게도 2, 3번의 경우 R이라는 언어에서 탁월한 기능으로 제공하고 있다. 물론 1번의 경우는 data.gov, UCI Machine Learning Repository(`http://archive.ics.uci.edu/ml/`) 등과 같은 오픈 데이터 사이트에서 구할 수 있으며, 몇몇 유명한 데이터는 R에서도 제공하고 있으니 큰 걱정은 하지 않아도 된다. 4번의 경우 많은 분들이 비싼 툴인 일러스트레이터를 주로 활용하지만 필자의 경우 오픈소스인 Inkscape로 간단하게 편집하곤 한다.
이런 이유로 정리된 이후 강좌 차례는 아래와 같다.

1. R로 하는 데이터 시각화의 시작 (금번 강좌)
2. R 기본 문법 및 사용법
3. R로 데이터 다루기(data munging with R) (data.table, plyr, sqldf 패키지 비교/이용)
4. ggplot2를 이용한 R 시각화
5. Inkscape를 활용한 그래프 후처리

위 강좌를 따라오기 위해 필요한 사전 지식은 프로그래밍 언어에 대한 경험뿐이다. 프로그래밍 경험이 없다고 하더라도 R을 첫 프로그래밍 언어로 배우기엔 손색이 없으니 의지만 있으면 가능할 것이라 생각한다.

[4] `http://flowingdata.com/2010/09/28/poll-results-what-do-you-use-to-analyze-andor-visualize-data/` 에 자세한 설명을 하고있음

Chapter 2

R 프로그래밍

이 장의 목적은 R 언어(R Core Team, 2012)에 대한 소개와 더불어 프로그래밍 문법적인 부분을 중점으로 설명하고자 한다. 앞으로 긴 여정을 진행하기 전에 가장 중요한 준비물 중에 하나이므로 정확히 이해하고 넘어가길 바란다. R은 AT&T에서 개발된 S 언어의 오픈소스 버전으로 약 20년이 넘는 역사를 가진 통계 컴퓨팅 (statistical computing) 언어이며 데이터 시각화를 위한 매우 훌륭한 환경이다.
대표적인 함수형 언어인 Lisp에서 파생된 언어 특징을 포함하고 있으며, 그와 동시에 절차적인 문법구조도 지원하며 모듈화된 패키지 개발을 위해 객체지향적인 요소들도 포함하고 있다. 특히나 함수형 언어의 특징 덕분에 간결한 코드를 구현할 수 있으며, 병렬화 처리 전환이 쉬운 장점이 있다.
게다가 인터렉티브 쉘을 제공하고 있어 코드의 결과를 바로 실행해 확인해 볼 수 있으며 이 때문에 정확하고 빠른 프로그래밍이 가능한 장점을 가지고 있다. 인터렉티브 쉘의 활용을 통해 기존 함수를 고치거나 새로운 함수를 제작하여 반복적인 작업을 수행하게 하는 루틴화를 쉽게 한다.
R은 데이터를 다루는 언어다. 데이터 획득과 조작 (munging[1]), 모델링, 시각화 등은 데이터 분석을 위한 큰 줄기에 해당하는 업무들이다. R은 분석 업무를 구석구석 최고로 다뤄줄 수 있는 툴과 같은 존재이자 통합 개발 환경 (IDE) 으로 볼 수 있다. 그 환경을 이해하려면 R 언어의 문법을 알 필요가 있다. 사실 R은 주변의 많은 프로그래밍 언어에서 영향을 받았다. 특히 Lisp라는 함수형 언어에서 영향을 받았는데, 이 때문에 많은 R 초보자들이 배움에 어려움을 겪고 있다. 하지만 이 덕분에 데이터 분석 시 복잡하고 이해하기 힘든 코드를 사용할 필요가 사라졌다.
인터랙티브하게 분석한다는 것은 분석툴의 유연성을 기반으로 한다. 분석 시 사용한 코드는 다른 분석에서 재사용할 수 있어야 하며, 몇 번이고 유사한 분석을 큰 노력 없이 반복적으로 수행 가능해야 한다. 게다가 분석 데이터의 다양성 때문에 통계량으로 데이터를 이해하는 것

[1]사전에 나오지 않은 신조어로 데이터를 분석 목적에 맞게 조작을 하는 행위를 의미한다. '멍잉' 이라고 읽는다.

이상으로 시각화로 데이터를 이해하는 방법이 보편화됐다. ‘백 번 듣는 것보다 한 번 보는 게 낫다’는 말처럼, 나열된 수치보다 보기 쉽게 시각화한 결과가 훨씬 빠르고 확실한 상황 판단이 가능하기 때문이다.
이런 인터렉티브한 분석을 수행하기 위해 가장 적합한 환경 중 하나가 R이다. R은 수많은 소스에서 데이터를 가져올 수 있고, 최신의 다양한 알고리즘을 적용 가능하며, 언어적으로 재활용성을 중시해 CRAN(Comprehensive R Archive Network)이라는 패키지 배포 시스템을 갖고 있다. 게다가 분석가들이 가장 많이 사용하는 그래픽 라이브러리들을 포함하고 있다.

2.1 인터렉티브 분석(interactive analysis)과 R

하둡(Hadoop)과 같은 분산 컴퓨팅 환경이 없을 때, 메인프레임 같은 대용량 컴퓨터에서 데이터 분석 비용은 매우 비쌌다. 때문에 마음 놓고 분석해볼 수 있는 여건이 아니었다. 따라서 컴퓨팅 환경과 분석 파라미터를 신중하게 선택/입력해야 했고, 그 결과 파일도 수백 페이지에 이르렀다. 이런 결과 파일에서 원하는 내용을 선별하고 나머지는 버렸다. 대부분의 통계 함수나 패키지들은 이런 대형 컴퓨터 환경일 때 개발됐고, 현재까지 그런 분석 결과를 보여주는 방식을 따르기도 한다.
하지만 개인용 컴퓨팅 환경이 일반화되기 시작하면서 예전처럼 한번에 설정하고 엄청난 양의 분석 결과를 얻는 그런 과정에서 벗어나기 시작한다. 다양한 데이터 입력, 변환, 무응답 대체(data imputation), 데이터 추가 및 삭제, 시각화, 모델링 과정 모두를 인터랙티브하게 수행하면서 분석가 자신이 갖고 있던 데이터에 대한 질문들에 대한 답을 완전히 얻을 수 있는 방법이 일반화됐다.
인터랙티브하게 분석한다는 것은 분석툴의 유연성을 기반으로 한다. 분석 시 사용한 코드는 다른 분석에서 재사용할 수 있어야 하며, 몇 번이고 유사한 분석을 큰 노력 없이 반복적으로 수행 가능해야 한다. 게다가 분석 데이터의 다양성 때문에 통계량으로 데이터를 이해하는 것
이상으로 시각화로 데이터를 이해하는 방법이 보편화됐다. ‘백 번 듣는 것보다 한 번 보는 게 낫다’는 말처럼, 나열된 수치보다 보기 쉽게 시각화한 결과가 훨씬 빠르고 확실한 상황 판단이 가능하기 때문이다.
이런 인터렉티브한 분석을 수행하기 위해 가장 적합한 환경 중 하나가 R이다. R은 수많은 소스에서 데이터를 가져올 수 있고, 최신의 다양한 알고리즘을 적용 가능하며, 언어적으로 재활용성을 중시해 CRAN(Comprehensive R Archive Network)이라는 패키지 배포 시스템을 갖고 있다. 게다가 분석가들이 가장 많이 사용하는 그래픽 라이브러리들을 포함하고 있다.

그림. 2.1: 윈도우용 R 다운로드 페이지

2.2 R을 설치하자

R은 `http://r-project.org`에서 내려받을 수 있다. R 자체뿐만 아니라 패키지라는 라이브러리와 같은 파일덩어리들은 CRAN이라는 곳에서 관리한다. Perl언어의 다운로딩 가능 리소스가 CPAN(Comprehensive Perl Archive Network)에서 일괄 관리되는 것과 같다. 따라서 R을 설치하기 위해서는 첫 페이지 왼쪽 메뉴에서 CRAN링크를 클릭한다. 이어서 'Korea' 항목으로 이동해 `http://cran.nexr.com`을 선택하면, 각 운영체제별로 컴파일된 바이너리 또는 소스를 다운 받을 수 있다. 국내 통계 자료로 보자면 R 사용자의 90% 이상이 윈도우 운영체제를 사용하고 있으므로 앞으로 예제는 윈도우 기반으로 소개한다.

계속해서 'Download R for Windows'를 클릭해 나타나는 창에서 'base'를 선택해 R 바이너리를 내려 받는다. 2013년 2월 4일 글을 쓰는 시점의 윈도우 바이너리 다운로드 화면은 그림2.1과 같다.

사실 `http://cran.nexr.com`은 필자가 관리하고 있고, 넥스알에서 후원하고 있는 서버 리소스다. 하루에 두 번 메인 서버와 싱크하기 때문에 대부분의 경우 가장 최신의 파일을 담고 있다. 많은 사용자가 64비트운영체제(OS)를 사용하는데, 이 바이너리를 설치할 때 x64 바이너리만 선택해 설치하기 바란다. 사실 64비트 OS에서는 x64바이너리가 가장 최적으로 작동하기 때문이다. 특히 운영체제에서 4GB이상의 메모리를 사용한다면 반드시 x64바이너리를 사용해야 한다. 물론 x86 바이너리를 함께 설치해도 작동에는 문제될 게 없지만, 사용하지 않을 바이너리까지 설치하면 시스템이 복잡해지고 공간낭비까지 불러온다. 설치과정에서는 '32-bit Files'을 체크 해제하면 x64 바이너리만 설치된다. 물론 32비트 OS를 사용한다면 64비트 바이너리 선택

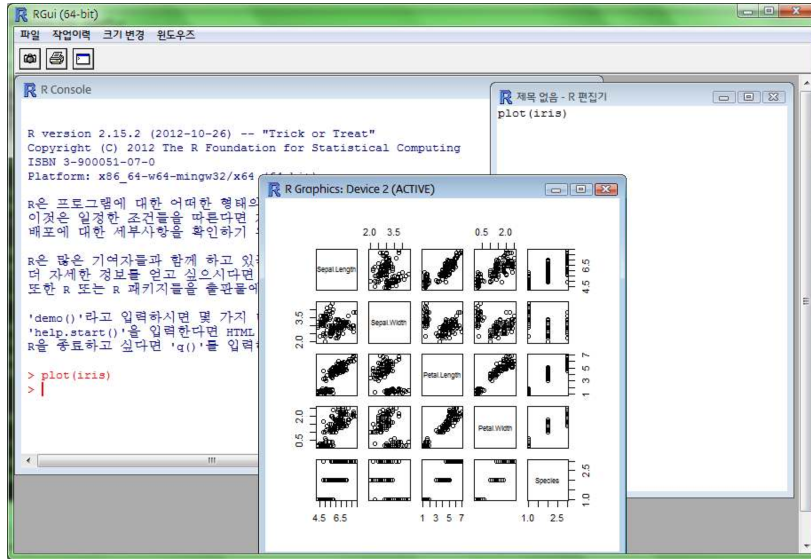

그림. 2.2: 윈도우용 R 실행 화면

화면은 나오지도 않는다.
소스코드를 통해 직접 빌드해 보는 것도 R이 어떻게 동작하는지 이해할 수 있는 좋은 방법이지만, 그 과정이 간단하지 않고 본 장의 주제와 직결되지 않아 여기서는 생략한다. 하지만 호기심 많은 독자라면직접 한번 시도해 보기 바란다[2].

2.3 R GUI와 IDE환경 소개

그림2.2는 일반적인 윈도우 R 콘솔화면을 보여준다. R을 설치한 후에 실행한 화면을 보면 썰렁하기 그지 없다. 단순히 코드에디터와 인터렉티브 쉘 그리고 가끔 팝업이 되는 help페이지나 그래프창이 전부이기 때문이다. 이 때문에 R을 처음 사용하는 사용자들이 어려워한다. 따라서 좀더 편리한 사용 시스템이 없는지 찾아보는 시간을 가져 보는것도 좋을거 같다.
일단 R을 손쉽게 사용하게 하기 위한 공개된 몇가지 IDE(Integrated Development Environment)와 GUI(Graphical User Interface)환경을 나열해 보도록 하겠다.
GUI환경은 아래와 같다.

- Poor Man's GUI (pmg)

[2]R은 데비안 계열의 리눅스에서 개발이 진행된다. 따라서 리눅스 계열에서 가장 잘 동작한다. 특히 우분투 계열에서는 특별한 설정이 없이 'sudo apt-get install r-base'과 같은 명령어로 손쉽게 R바이너리 설치가 가능하다. 레드헷 계열도 yum으로 설치가 가능하나 별도의 저장소(repository)를 설정해 줘야 된다. 리눅스 사용에 꺼리낌이 없는 사용자라면 리눅스에서 R을 사용하는 방법을 추천한다. 이는 R 코딩 용이성 측면에서 뿐만 아니라 리눅스 서버를 사용할 시 꽤 쾌적하게 서버 리소스를 사용할 수 있는 자유로움도 주기 때문이다.

- Jaguar : Java GUI for R
- R commander
- Rattle GUI

이중에서 가장 널리 사용되는 GUI는 R commander(Fox, 2005)로 설치 방법은 아래와 같다.

```
> #설치
> install.packages("Rcmdr")
> #실행
> library(Rcmdr)
```

실행을 시키는 중에 여러 패키지 설치가 수행될 수 있으며 이 과정에서 시간이 좀 걸릴 수 있다.
R실행 환경에 익숙치 않다면 메뉴 방식으로 명령어를 생성해주는 이와 같은 GUI를 기반으로 초반접근을 해보는것도 나쁘지 않다. 하지만 이런 툴을 사용하면서 명심해야 될 부분은 실제 명령어가 어떻게 생성되는지 확인하고 기억해야 된다는 것이다. 그런 과정을 통해서 R이 동작하는 방식을 이해하게 되며 좀더 유연하게 R을 사용할 수 있기 때문이다.
Rattle은 데이터 마이닝에 특화된 환경을 제공하고 있으며, 업무가 데이터 마이닝이라면 이 환경을 한번정도 사용해볼 필요가 있다. 이 툴을 기반으로 데이터마이닝을 설명한 도서(Williams, 2011)가 있는데 툴설명과 더불어 R을 기반으로 데이터마이닝을 하는 좋은 예제들이 있는 책이니 관심 있는 독자분은 읽어보는 걸 추천한다.

```
> #설치
> install.packages("rattle")
> #패키지 로딩
> library(rattle)
> #실행
> rattle()
```

GUI환경에서 R명령어들이 익숙해 진 후라면 적절한 IDE를 구하게 되는데, 현재 많이 쓰이는 IDE환경은 아래와 같다.

- RStudio(RStudio Team, 2012)
- RKward
- Eclipse with StatET
- Tinn R

상용 R IDE로 Revolution R[3]이 있고 학생의 경우 아카데미 버전을 무료로 제공하고 있으니 관심 있는 분들은 사용해 봐도 좋다.

2.4 RStudio를 이용하자

물론 처음부터 R 기본 쉘을 사용하는 것도 나쁘지 않지만, RStudio(`http://www.rstudio.org/`)를 사용하면 가장 최적의 R사용 환경을 유지할 수 있기 때문에 정말 편하다. RStudio는 그 자체만으로도 많은 기능을 포함하고 있어서 이곳에서 전체를 소개하기는 쉽지 않다. 일단 필자가 아는 RStudio의 장점은 다음과 같다.

- 에디터, 콘솔, 명령어 히스토리, 시각화, 파일탐색, 패키지 관리 등을 한 화면에서 보여준다.
- 프로젝트의 관점으로 파일 관리를 해준다. 물론 소스코드 관리 시스템과 연계할 수 있다.
- 빌트인 데이터 뷰어 내장, 플로팅 히스토리, R help 결합, Sweave(Leisch, 2002), knitr(Xie, 2013) 통합
- R Markdown(Allaire *et al.*, 2013) 내장으로 문서와 코드를 결합할 수 있게 하고, 재현성 있는 분석을 가능하게 함
- 패키지 빌드 자동화와 Rcpp(Eddelbuettel and François, 2011) 편집 환경 제공
- 리눅스, 맥, 윈도우 등 멀티 플랫폼 지원

RStudio는 무엇보다 현재로서 가장 인기 있는 IDE다. 게다가 클라이언트/서버 환경으로 설정해두면 풍부한 서버 리소스를 원격에서 활용할 수 있다. 특히 웹 브라우저 자체가 편집기가 되어 세션이 죽어도 최신의 작업 히스토리와 환경을 간직하고 있어 브라우저만 있으면 어느 곳에서든지 자신이 작업한 환경을 그대로 사용할 수 있다. 필자는 회사의 128GB 메모리 서버에 RStudio를 설치해두고 리소스가 풍부하지 않은 데스크톱에서 원격으로 접속해 사용하는데 이는 정말 최고의 작업 환경이다[4].

---

[3] `http://www.revolutionanalytics.com/products/revolution-r.php`에서 배포한다. MS의 컴파일러로 컴파일을 한 R 환경을 제공하며 회사 소개자료에서는 오픈소스 R보다 다소 빠른 속도를 보여준다고 한다.

[4] R은 모든 데이터를 메모리에 올려놓아야만 분석이 가능하다. 일단 메모리에 데이터를 올리려면 시스템 메모리가 충분해야 된다. 게다가 대부분의 분석 업무는 같은 데이터를 쪼개고 붙이고 하는 작업들의 연속인 관계로 필요 메모리 량은 더 늘어나게 마련이다. 따라서 필자는 메모리에 대한 고민을 업무를 하면서 하고 싶지 않아 될 수 있는한 최대 스펙의 서버에서 분석을 수행한다. 단순히 메모리만 많다고 좋은것도 아니다. 만일 큰 데이터를 멀티코어프로세싱으로 처리하고 싶다고 한다면 CPU 스펙도 메모리 못지 않게 좋아야 된다. 멀티코어프로세싱에 대한 내용도 이후에 정리를 하도록 하겠다.

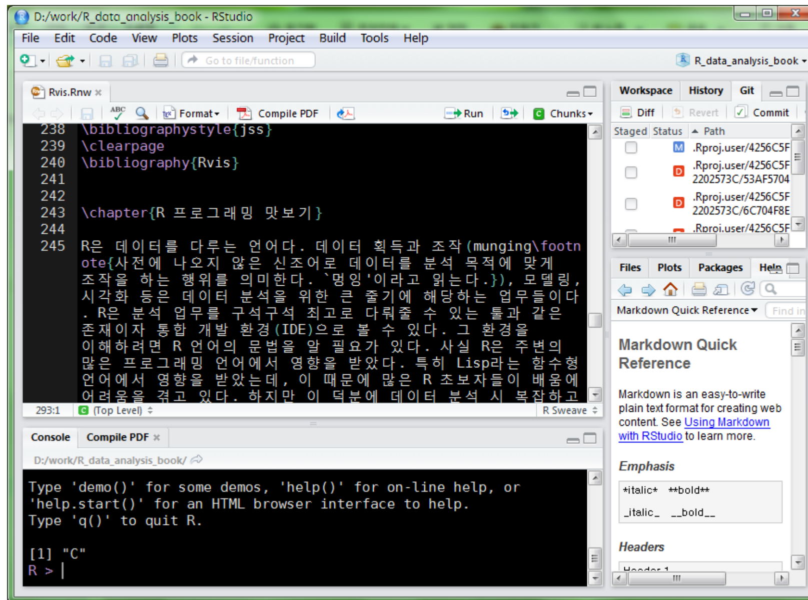

그림. 2.3: 윈도우용 RStudio 화면 예

2.5 R 프로그래밍

패키지 설치 및 도움 얻기

이미 앞에서 명령어를 사용해 봤겠지만 패키지를 관리하는 방법에 대해서 알아볼 필요가 있다. 명령어 설명 이전에 CRAN 미러링 서버가 적절하게 설정이 되었는지 확인이 필요하다. 이 미러링 서버는 중앙 CRAN서버의 모든 데이터를 복사해 지리적으로 가까운 사용자에게 쾌적하게 패키지 관리를 할 수 있게 하는 서버이다. 글을 쓰고 있는 2013년 2월 현재 한국에서 Nexr과 중앙대학교에서 서버를 제공하고 있다. RStudio에서 Tools-Options-Packages 메뉴에 들어가서 CRAN mirror[5]가 NexR이나 중앙대학교로 설정이 되어 있는지 확인하고 설정하면 미러링 서버로 해당 서버가 사용되게 된다. 만일 이 부분이 적절하게 설정되지 않았다면 패키지 다운로딩과 설치에 다소 시간이 걸릴 수 있다.

예를 들어 필자가 만든 KoNLP(Jeon, 2012)라는 한글 텍스트 분석 패키지를 설치하고 싶다면 아래와 같은 명령어를 실행하면 미러링 서버에서 해당 패키지를 다운로드 받아 설치까지 자동으로 수행되게 된다.

[5] `http://cran.r-project.org/mirmon_report.html`에서 미러 서버가 얼마나 업데이트를 잘 하고 있는지 확인 가능하다. 만일 패키지 설치나 관리가 생각한대로 동작하지 않는다면 자신의 미러서버가 현재 잘 동작하고 있는지 이 페이지에서 확인해 볼 수 있다.

```
> install.packages("KoNLP")
```

현재 패키지가 어떤 것들이 설치 되어 있는지 확인하고 싶으면 RStudio 오른쪽 메뉴창에서 'Packages' 항목을 선택해 리스트를 확인해 보면 되고, 콘솔창에서는 아래와 같은 명령어를 입력하면 리스트를 출력해 줄 것이다.

```
> installed.packages()
```

R의 전체 패키지는 글을 쓰는 현재 4000여개가 넘기 때문에 각종 패키지들의 업데이트가 자주 있는 편이다. 따라서 주기적으로 패키지 업데이트를 해줄 필요가 있는데 RStudio에서는 'Tools-Check for Package Updates'를 클릭해 업데이트 여부를 체크하여 업데이트 하면 되고 명령어로는 아래의 명령어로 체크하면 업데이트 가능한 패키지들을 리스트업 해준다.

```
> old.packages()
```

체크와 더불어 업데이트까지 설치하는 명령어는 아래와 같다.

```
> update.packages()
```

때로는 여러분들이 책에서 apriori 알고리즘을 공부하고 R에서 구현체를 찾아볼 생각을 할 수도 있을 것이다. 그럴때 이 함수나 구현체가 어디에 존재하는지 알아보기 위해서 고민을 할 수 있을텐데 R에서는 이런 함수나 구현체를 찾기 위한 여러 명령어들을 제공하고 있다.

```
> #help 시스템에서 해당 단어를 찾아준다.
> help.search("apriori") # 혹은 ??apriori
```

위 명령어는 같은 역할을 하지만 한가지 단점이 해당 단어가 현재 R help 시스템에 존재해야 된다. 쉽게 이야기 하면 사용자의 로컬 R 시스템에 해당 구현체가 있어야만이 된다는 것이다. 만일 없다면 아무 검색 결과가 나오지 않을 것인데, 이럴 경우에 사용할 수 있는 명령어가 RSiteSearch() 명령어인데, `http://search.r-project.org`의 검색 결과를 리턴해준다.

```
> RSiteSearch("apriori")
```

RSiteSearch() 함수의 일부 기능을 하는 함수로서 `http://search.r-project.org/`의 결과를 좀더 정리된 형태로 웹브라우저를 통해 보여주는 sos 패키지(Graves *et al.*, 2012)도 있다. 하지만 자세한 결과를 보고 싶다면 RSiteSearch() 함수를 사용하는 걸 추천한다.

```
> install.packages("sos")
> library(sos)
> findFn("apriori") # 혹은 ???apriori
```

패키지를 로딩했으나 어떤 함수가 있는지 궁금할때는 아래 명령어를 사용한다[6].

```
> help(package="arules")
```

R 기본 연산

R을 실행하면 R콘솔이 뜨면서 메시지와 더불어 "> "와 같은 콘솔 프롬프트가 뜨게 된다. 대부분의 인터프리터 언어의 경우 이런 형태를 띄고 있으며 이 문자 뒤에 명령어를 입력하고 엔터키를 누르면 해당 라인에 있는 명령어를 수행하고 바로 평가 결과를 리턴하게 된다.

```
> 1 + 2 + 3

## [1] 6

> 1/2

## [1] 0.5

> (1 + 2) * 3

## [1] 9
```

각 명령 결과의 머리를 보면 '[1]'이라는 문자가 함께 출력이 되는데, 이 문자는 R에서 사용하는 벡터(vector)라는 자료형의 길이의 인덱스를 의미한다. 벡터는 R에서 가장 기본이 되는 자료형으로서 '3'이라는 숫자하나도 길이가 1인 벡터형으로 인식을 해 처리된다. 위의 결과들은 모두 길이가 하나인 결과를 리턴함으로 '[1]'이라는 인덱스 번호만 뜨게 된다. 이 인덱스 번호는 출력한 벡터 결과의 크기를 가늠하거나 특정 인덱스 번호를 알아야 될 경우 꽤 유용하다.

```
> 1:100

##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28
##  [29]  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56
##  [57]  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84
##  [85]  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

':'는 첫번째 숫자에서 시작하고 마지막 숫자로 끝나는 숫자형 벡터를 리턴해 준다. 위 결과에서는 연속된 값을 출력해서 인덱스 번호를 유추하기 쉬우나, 다소 복잡한 결과물일 경우 이런 인덱스 번호는 굉장히 편리한 장점을 준다. 보는 바와 같이 결과 각 라인의 첫번째 인자의 인덱스 번호를 출력해 준다.

[6] arules(Hahsler *et al.*, 2012)는 apriori 알고리즘을 구현하고 있는 패키지이다.

```
> c(1,2,3,4,5)

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

위와같이 긴 벡터를 생성하기 위해서는 'c'ombine 함수를 사용하면 된다.

```
> c(1,2,3,4,5) + c(5,4,3,2,1)

## [1] 6 6 6 6 6

> c(1,2,3,4,5) * c(5,4,3,2,1)

## [1] 5 8 9 8 5

> c(1,2,3,4,5) / c(5,4,3,2,1)

## [1] 0.2 0.5 1.0 2.0 5.0

> c(1,2,3,4,5) - c(5,4,3,2,1)

## [1] -4 -2  0  2  4
```

길이가 같은 벡터간의 연산은 우리가 알고 있는 수학의 벡터 연산과 같은 방식으로 각 벡터의 인덱스 쌍 사이의 연산을 리턴하고 이를 다시 벡터로 만들어 같은 길이의 벡터가 리턴된다.
만일 길이가 다르다면 어떻게 될까?

```
> 1 + c(1,2,3,4,5)

## [1] 2 3 4 5 6

> c(1,2) + c(1,2,3,4,5)

## Warning: 두 객체의 길이가 서로 배수관계에 있지 않습니다

## [1] 2 4 4 6 6

> c(1,2) + c(1,2,3,4,5,6)

## [1] 2 4 4 6 6 8
```

위의 결과를 보고 예측을 해보자!
R은 길이가 다른 벡터간의 연산은 길이가 적인 벡터를 반복해서 사용해 연산을 하고 결과를 리턴한다. 위의 두번째 결과는 에러가 나는데 이는 긴 길이의 벡터가 작은 벡터 길이의 배수가 아니기 때문이다. 그러나 마지막 결과는 정확하게 도출 됨을 볼 수 있다. 이런 방식의 연산방법은 R을 사용하면서 상당히 많은 부분에서 편리하게 활용이 되니 기억해 두기 바란다.
숫자 뿐만 아니라 문자열도 벡터에 저장된다.

```
> #   '#' 이후의 문자열은 주석입니다.
> "안녕하세요. 고감자입니다."

## [1] "안녕하세요. 고감자입니다."

> c("안녕하세요", "고감자입니다.")

## [1] "안녕하세요"     "고감자입니다."
```

함수 (Function)

R에서 실제 작업을 수행하는건 함수이다. 이미 이전 코드에서 수많은 함수를 사용해 왔으며, 프로그래밍에 익숙한 독자라면 함수가 어떤 형태로 구성되는지 이미 익숙해 졌을 것이다.

```
> foo(arg1, arg2, ...)
```

항수의 형태는 위와 같은 형태를 띄며 'arg1'은 함수에 넘겨지는 첫번째 인자, 'arg2'는 두번째 인자이다.

```
> sqrt(2)

## [1] 1.414

> abs(-24)

## [1] 24
```

'sqrt' 함수는 제곱근을 구하는 함수이며, 반환값으로 입력 인자로 넘겨진 숫자의 제곱근을 반환한다. 'abs'는 입력값의 절대값을 반환하는 함수이다.

좀더 자세히 내려가면 '+,-' 등의 연산자의 경우도 함수의 일환이다.

```
> 1+2

## [1] 3

> `+`(1,2)

## [1] 3
```

위 두 예를 제외하고 생각보다 많은 부분들이 함수로 이루어져 있으며 다만 우리가 사용하는 표현은 대부분이 달콤한 문법 (syntactic sugar) 들이다. 사실 위 2.5에서 설명한 연산자들 모두 함수들인데, 그만큼 R의 많은 부분들이 함수로 이뤄져 있다.
함수를 만드는 방법을 소개한다.

```
> foo <- function(arg1, arg2, ..., bar=bar_val){
+   print(arg1)
+   print(arg2)
+   print(1231232323131231,...)
+   print(bar)
+   return(1)
+ }
>
> ret <- foo("첫번째", "두번째", 5, bar="박")

## [1] "첫번째"
## [1] "두번째"
## [1] 1.2312e+15
## [1] "박"

> ret

## [1] 1
```

함수는 대부분 위와 같은 형태를 띈다. 'arg1, arg2'의 경우 첫번째 인자 두번째 인자를 의미하며 '...'는 대부분 함수 내부에서 다른 함수를 호출할때 해당 함수에 추가적인 인자를 넘겨주기 위해서 사용한다. '...' 이후에는 반드시 명명된 변수만이 올 수 있다. 위에서는 'bar='와 같이 인자를 넘겨줄 때 이름을 명명해서 넘겨줘야만 된다.

변수(Variables)

객체에 값을 저장하기 위해서 R콘솔에서는 '<-(혹은 ->)' 연산자를 사용한다. 물론 다른 언어에서는 '=' 연산자를 주로 사용하나 이 책에서는 '같다(==)'라는 의미와 혼동을 할 수 있어서 '<-'를 주로 사용하도록 하겠다. 단 함수에서 인자에 값을 넣을때는 '<-'는 동작하지 않으며 '='를 사용해야 된다.

```
> x <- 1
> x

## [1] 1
```

첫 라인은 "'x' 변수는 1을 얻었다(gets)"라고 이해하면 되며, 두번째 라인은 "x에 어떤 값이 들어 있는지 평가한다"라고 이해하면 된다. x는 그 자체로 심볼(Symbol)이며 1이라는 객체(Object)를 지칭하고 있다.

```
> x <- 3
> y <- 5
> z <- c(x,y)
> #z
```

```
> y <- 200
> z

## [1] 3 5
```

'z'에 저장되는 순간 값이 복사되며 이후에 'y'값이 변경이 되더라도 'z'값은 그대로 유지된다. 이유는 'c'함수가 평가될 당시 x,y값이 복사되어 z값으로 리턴되기 때문이다. R함수는 기본적으로 값에의한 호출(call by value)을 적용하고 있으며 별도로 트릭을 이용해 참조에 의한 호출(call by reference)을 구현할 수 있다. 만일 참조에 의한 호출을 하고 싶다면 R Envirement에 대한 이해가 있어야 되며, 공식적으로 지원하지 않는 기능이기 때문에 다소 교묘(tricky)하지만 이후에 간략하게 소개하도록 하겠다.

벡터(Vector)

이미 벡터는 지금까지 심심치 않게 사용해 왔다. 벡터는 단 한 종류의 자료형만을 저장할 수 있는 자료구조이다. 이 자료형들에는 정수형(integer), 부동소수점수(floating-point number), 복소수(complex number), 문자열(text), 논리값(logical value), 로(raw)등이 올 수 있다.
벡터에는 한가지 자료형만 존재할 수 있기 때문에 아래와 같이 자료형강제변환(coerces)이 수행되기도 한다. 아래 'typeof()'함수는 변수의 저장타입을 리턴한다.

```
> v <- c(1,2,3,4,5,6)
> v
> typeof(v)
> coerced <- c(1,2,3,4,5,6, "고감자")
> coerced
> typeof(v)
```

이런 변환은 데이터의 손실이 없는 가장 일반화된 자료형으로 변환된다.
그럼 벡터형 변수를 정의하고 특정 값들을 가져올 수 있는 여러 방법을 소개한다.

```
> vec <- c(1,2,3,4,5,6,7,8,9,10)
> #첫번째 원소
> vec[1]

## [1] 1

> #첫번째 원소를 다른 숫자로 대체
> vec[1] <- 100
> #홀수 인덱스에 있는 값들만
> vec[c(TRUE, FALSE)]

## [1] 100   3   5   7   9
```

```
> #첫번째부터 세번째 값
> vec[1:3]

## [1] 100   2   3

> #3의 배수
> vec[vec %% 3 == 0]

## [1] 3 6 9

> #3의 배수가 아닌것들
> vec[!(vec %% 3 == 0)]

## [1] 100   2   4   5   7   8  10

> #세번째, 열번째 인자를 제거한 벡터
> vec[-c(3,10)]

## [1] 100   2   4   5   6   7   8   9
```

위와 같은 특정 데이터 추출 방법은 R기반 데이터 멍잉(munging) 전반에서 사용되기 때문에 반드시 익혀둬야 된다.
두 벡터를 결합하는 방법은 아래와 같다.

```
> v <- 1:10
> z <- 10:1
> comb <- c(v,z)
> comb2 <- append(v,z)
> comb == comb2

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

':'가 시작숫자에서 끝 숫자까지 1씩 증가하거나 감소하는 숫자형 벡터를 반환해주는데, 때로는 증가방식을 다르게 하고 싶을때가 있을 것이다.

```
> seq(from=1,to=10, by=2)

## [1] 1 3 5 7 9
```

벡터의 길이는 'length()'함수를 통해 알 수 있다.

```
> v <- c(1:20)
> length(v)

## [1] 20

> length(v) <- 30
> v
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 NA NA NA NA NA NA NA NA NA NA
```

```
> length(v) <- 10
> v
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

만일 실제 데이터 길이보다 더 긴 길이 정보를 넣을 경우 나머지 슬롯에는 'NA'(Not Available) 값이 들어가게 된다. 'NA'는 값이 없다는 것을 의미하며 데이터 분석업무에서는 주로 손실된 값을 지칭한다. 그러나 실제 벡터 길이보다 작은 값을 길이 정보로 넣을 경우 길이가 넘는 범위에 있는 데이터는 제거된다.
때때로 미리 길이가 결정된 벡터를 만들 필요가 있다. R과 같은 인터프리터 언어들은 대부분 두 벡터를 결합하거나 원소 하나를 추가하는 작업을 수행할때 매번 새로운 벡터를 만들게 된다. 내부적으로는 메모리 영역을 할당하고 초기화를 하는등 많은 부가적인 연산이 들어가게 되는데, 이런 작업이 많을때는 미리 사용할 충분한 크기의 벡터를 만들어두고 인덱스번호로 접근해 값을 바꾸기만 하면 된다. 이럴때 미리 정의된 벡터를 만드는게 필요하다.

```
> v <- vector(mode = "numeric", length = 10)
> v
```

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

```
> v[1] <- 1
> v
```

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

리스트 (List)

R에서 리스트는 벡터와 유사하게 순서화된 객체들을 포함하는 자료형이나, 한가지 데이터형만을 포함하는게 아닌 여러 종류의 데이터 객체를 포함할 수 있는 장점을 가지고 있다.

```
> l <- list(1,2,'3',4,5,6)
> l[3]
```

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

```
> l[[3]]
```

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

```
> typeof(l[3])
```

```
## [1] "list"

> typeof(l[[3]])

## [1] "character"
```

위와 같이 숫자형 뿐만 아니라 문자형 데이터를 섞어서 저장이 가능하며 벡터와 같이 인덱스 번호로 내부 데이터 접근이 가능하다. 그러나 벡터와 같이 '[]'로 접근할 경우 리스트형을 리턴하게 되며 '[[]]'으로 접근하면 원소의 자료형 그대로 리턴하게 된다.

```
> l <- list(name="고감자", age=35, sex="남자")
> l$name

## [1] "고감자"

> l["name"] # 리스트형 반환

## $name
## [1] "고감자"

> l[["name"]] # 문자형 반환

## [1] "고감자"

> l[1] # 리스트형 반환

## $name
## [1] "고감자"

> l[[1]] == l$name

## [1] TRUE

> l[2:3] #리스트형 반환

## $age
## [1] 35
##
## $sex
## [1] "남자"

> #에러 출력
> #l[[2:3]]
```

위와 같이 C언어의 구조체 형식의 자료형을 구성할 수도 있으며 각 원소에 접근하는 방법은 위의 예를 통해서 알 수 있다.

```
> l$address <- "서울 관악구"
> l

## $name
## [1] "고감자"
##
## $age
## [1] 35
##
## $sex
## [1] "남자"
##
## $address
## [1] "서울 관악구"

> l[[5]] <- "비고"
> l

## $name
## [1] "고감자"
##
## $age
## [1] 35
##
## $sex
## [1] "남자"
##
## $address
## [1] "서울 관악구"
##
## [[5]]
## [1] "비고"

> l[6:8] <- c(FALSE, TRUE, TRUE)
> l

## $name
## [1] "고감자"
##
## $age
## [1] 35
##
## $sex
## [1] "남자"
##
## $address
## [1] "서울 관악구"
##
```

```
## [[5]]
## [1] "비고"
##
## [[6]]
## [1] FALSE
##
## [[7]]
## [1] TRUE
##
## [[8]]
## [1] TRUE
```

원소를 추가하는 대표적인 방법은 위와 같다.

```
> l[[8]] <- NULL
> l$address <- NULL
> length(l)

## [1] 6
```

원소 삭제 방법은 원소에 NULL을 입력해주면 된다(삭제 후 리스트 원소 순서를 확인해 보길 바란다). 리스트의 길이는 벡터와 같은 방식으로 확인 가능하다.

배열(array)와 행렬(matrix)

때로는 다차원 벡터를 다룰 필요가 있는데 배열과 행렬은이를 위한 데이터 구조이다. 배열은 주로 3차원 이상의 데이터를 다룰때 사용되며 행렬은 수학에서의 그 행렬과 동일한 성격을 가지고 있는 데이터 구조이며 2차원의 벡터를 다루고 있다. 따라서 벡터와는 *dimension*이라는 속성의 유무만 다르고 같다.

```
> arr <- array(1:3, c(2,5,2))
> is.array(arr)

## [1] TRUE

> dim(arr)

## [1] 2 5 2

> # dim 속성을 없앤다.
> attr(arr, "dim") <- NULL
> dim(arr)

## NULL

> is.array(arr)
```

```
## [1] FALSE

> is.vector(arr)

## [1] TRUE
```

위 코드는 c(1,2,3) 벡터를 3차원으로 표현하는 방법이며, 숫자는 각 차원의 최고 인덱스 숫자를 의미한다. 20개의 벡터원소가 필요하며 이를 위해 c(1,2,3) 벡터는 반복적으로 생성되게 된다.
위에서 처럼 dim(dimension)속성을 없애버리면 이 데이터 구조는 배열이라 할 수 없게 되고 벡터가 된다.

```
> mat1 <- matrix(1:10)
> mat1

##       [,1]
##  [1,]    1
##  [2,]    2
##  [3,]    3
##  [4,]    4
##  [5,]    5
##  [6,]    6
##  [7,]    7
##  [8,]    8
##  [9,]    9
## [10,]   10
```

아무 인자를 주지 않고 값만 줘서 행렬을 만들면 열기준으로 원소가 배치되는 것을 볼 수 있다. 이는 행렬이 열 기반 표현방식을 기본적으로 따르고 있기 때문이다. 따라서 입력된 데이터와 더불어 사용자가 원하는 모습의 행렬을 만들기 위해서는 아래와 같이 표현하면 된다.

```
> mat2 <- matrix(1:10, nrow=2, ncol=5)
> mat2

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

아래와 같이 dim속성에 차원을 할당하면 행렬로 변환이 가능하다.

```
> # attr(mat1, "dim") <- c(2,5) 와 같은 동작을 수행한다.
> dim(mat1) <- c(2,5)
> mat1

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

```
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10

> mat1 == mat2

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

위의 벡터 연산에서 처럼 인덱스와 여러 연산을 통해서 부분의 데이터를 가져오고 삭제하고 변경하는 연산이 가능하다.

```
> mat <- matrix(1:10, nrow=2, ncol=5)
> mat[2,3]

## [1] 6

> mat[2,]

## [1]  2  4  6  8 10

> mat[,3]

## [1] 5 6

> mat[c(1,2), c(1,2,3)]

##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6

> mat[-c(1),]

## [1]  2  4  6  8 10

> mat

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

이전의 벡터 삽입, 삭제 연산에 대해서 충분히 이해를 했다면 위 결과들도 쉽게 이해가 가능할 것이다.

필요한 열이나 행을 추출할 때 예상한대로 동작하는 않는 경우가 몇가지 있는데 이중에 하나를 소개한다.

```
> mat[,1]

## [1] 1 2

> mat[,c(1,2)]

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

위 두 결과의 차이 중에 하나는 첫번째 것은 벡터를 리턴한다는 것과 마지막은 행렬을 리턴한다는 것이다. 이 두 차이가 뭐 큰 것이냐 할 수 있지만 만일 스크립트화 된 코드를 수행하게 된다면 이 두 구문의 차이를 두지 않고 개발을 했을 경우 처리 도중 에러를 낼 수 있다. 예를 들어 어떤 조건에 맞는 열을 뽑아서 두 열을 결합한다고 했을 경우 두 데이터 구조가 맞지 않으면 원하는 연산 결과를 얻기 힘들 것이다. 이를 위해 단 하나의 열이나 행을 리턴하더라도 원래 데이터 구조를 기반으로 리턴하게 하는 방법이 필요하다.

```
> mat[,1, drop=FALSE]

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

위와 같은 구문을 사용하게 되면 반환되는 데이터 구조도 행렬이 된다[7].

```
> rownames(mat) <- c("첫번째","두번째")
> colnames(mat) <- c("first", "sencond", "third", "fourth", "fifth")
> mat

##        first sencond third fourth fifth
## 첫번째     1       3     5      7     9
## 두번째     2       4     6      8    10

> mat[,c("first", "fifth")]

##        first fifth
## 첫번째     1     9
## 두번째     2    10

> mat["첫번째",, drop=FALSE]

##        first sencond third fourth fifth
## 첫번째     1       3     5      7     9
```

위와 같이 열 이름과 행 이름을 지정할 수 있으며, 이름으로 특정 데이터에 접근이 가능하다.

[7]이는 데이터프레임(data.frame)에서도 동일하게 적용되는 부분이다.

데이터프레임 (Data Frames)

R에서 아마도 데이터프레임은 가장 널리 활용되는 데이터 구조이며, 행렬과 유사하지만 행렬과는 다르게 각열이 다른 데이터 타입을 가질 수 있다. 흡사 RDBMS의 테이블과 유사하다고 간주하고 접근하면 오히려 쉽게 접근이 가능할 것이다. 실제 SQL 쿼리문을 기반으로 데이터프레임을 조작하는 패키지[8]도 존재하며 이에 대해서는 추후에 설명하도록 하겠다.

```
> df <- data.frame(id=c("gogamza", "haven","heewon", "gamza"), math=c(30,21,70,80))
> names(df)

## [1] "id"   "math"

> dim(df)

## [1] 4 2

> str(df)

## 'data.frame': 4 obs. of  2 variables:
##  $ id  : Factor w/ 4 levels "gamza","gogamza",..: 2 3 4 1
##  $ math: num  30 21 70 80

> summary(df)

##        id          math
##  gamza  :1   Min.   :21.0
##  gogamza:1   1st Qu.:27.8
##  haven  :1   Median :50.0
##  heewon :1   Mean   :50.2
##              3rd Qu.:72.5
##              Max.   :80.0

> nrow(df)

## [1] 4

> ncol(df)

## [1] 2
```

코드에서 데이터프레임을 정의하는 일반적인 방법과 더불어, 데이터프레임을 조작하면서 나올 수 있는 'name(), dim(), str(), summary()'와 같은 함수의 결과를 보여준다. 이런 종류의 함수는 자신이 모르는 데이터프레임 형식의 데이터가 주어졌을 때 일반적으로 가장 먼저 수행시켜 보는 함수들이다.
각 원소에 대한 접근은 행렬과 같은 방식으로 수행된다.

[8]'sqldf'(Grothendieck, 2012)와 같은 패키지가 있다.

```
> df[1,2]

## [1] 30

> df[1,c(1,2)]

##        id math
## 1 gogamza   30

> df$math

## [1] 30 21 70 80

> #수학점수가 60 점 이상인 행
> df[df$math >= 60,]

##        id math
## 3 heewon   70
## 4  gamza   80

> #아이디가 gogamza인 행
> df[df$id == "gogamza",]

##        id math
## 1 gogamza   30
```

위와 같은 직관적인 방식의 데이터 추출 방식도 좋지만 만일 필드명이 길어지거나 할 경우에는 코드가 너무 길어지고 읽기 힘들어 지는 상황이 발생할 수 있다. 이를 위해 아래와 같은 함수를 사용하기도 한다.

```
> # df[df$math >= 60,] 와 같다
> subset(df, math >= 60)

##       id math
## 3 heewon   70
## 4  gamza   80
```

만일 데이터프레임의 한 컬럼에 특정 숫자를 더한 다음에 새로운 컬럼이나 이미 있던 컬럼의 값을 갱신할 경우를 생각해본다면 아래와 같은 형태가 될 것이다.

```
> #시험 문제가 잘못되어 모든 학생에게 1점씩 그냥 준다.
> df2 <- df
> df2[, "math"] <- df2[, "math"] + 1
> df2

##        id math
## 1 gogamza   31
```

```
## 2    haven   22
## 3   heewon   71
## 4    gamza   81
```

하지만 이것도 길어질 수 있어 필자의 경우는 아래와 같은 함수를 사용한다.

```
> transform(df, math = math + 1)

##        id math
## 1 gogamza   31
## 2    haven   22
## 3   heewon   71
## 4    gamza   81
```

'transform'함수는 데이터프레임의 특정 열이름을 기반으로 데이터를 접근하며 그 해당열에 특정 연산을 수행해 새로운 열이나 기존열에 값을 넣어주는 역할을 한다. 그러나 아래와 같이 이전 연산에 영향을 받아야만 되는 형태의 조작은 독자분들이 의도하는 바대로 결과를 주지 않는다.

```
> transform(df, math = math + 1, korean = math + 3)

##        id math korean
## 1 gogamza   31     33
## 2    haven   22     24
## 3   heewon   71     73
## 4    gamza   81     83
```

이를 위해 아래와 같은 'within'함수를 사용한다.

```
> within(df, {
+   math <- math + 1
+   korean <- math + 3
+   })

##        id math korean
## 1 gogamza   31     34
## 2    haven   22     25
## 3   heewon   71     74
## 4    gamza   81     84
```

'within'과 유사한 동작은 하는 함수로는 'plyr'(Wickham, 2011)패키지의 'mutate'함수가 존재하는데, 'within'함수가 새로운 열을 추가하는 동작을 수행함에 있어 추가 순서가 뒤섞이는 경향이 있는 반면에 'mutate'함수의 경우 그 순서를 지켜주며 결과물을 내준다[9].

[9]이 말이 어떤 뜻인지 이해가 되지 않는 독자들은 두 함수를 가지고 세가지 이상의 새로운 열을 추가하는 연산을 수행해 보길 바란다.

```
> names(df)[1] <- "이름"
> df

##       이름 math
## 1 gogamza   30
## 2   haven   21
## 3  heewon   70
## 4   gamza   80
```

위 명령어는 첫번째 열이름인 'id'를 '이름'으로 변경하는 명령어다.

```
> rbind(df, df)

##       이름 math
## 1 gogamza   30
## 2   haven   21
## 3  heewon   70
## 4   gamza   80
## 5 gogamza   30
## 6   haven   21
## 7  heewon   70
## 8   gamza   80

> cbind(df, df)

##       이름 math    이름 math
## 1 gogamza   30 gogamza   30
## 2   haven   21   haven   21
## 3  heewon   70  heewon   70
## 4   gamza   80   gamza   80
```

같은 열이름과 데이터 타입을 가진 데이터프레임들을 행기준으로 결합할 때 'rbind' 명령어를 사용하며, 열기준으로 결합을 할 땐 'cbind'를 사용한다[10].
R에는 분석 가능한 기본 데이터셋이 존재하여, 여러분들이 따로 데이터를 읽어들일 필요 없이 연습을 할 수 있게 해놓았다.

```
> #사용 가능한 데이터셋 목록
> data()
> #데이터 로딩하며 cars 데이터셋을 cars라는 이름의 데이터프레임으로 로딩한다.
> data(cars)
> #요약정보
> summary(cars)
```

[10] 이 명령어는 행렬에서도 동일하게 적용된다.

R에서의 객체지향적인 개념 간단 이해

R을 사용하다 보면, 'print()' 나 'plot()' 혹은 'predict()"과 같은 함수들이 전달되는 객체에 맞게 적절한 동작을 하는 것을 경험할 수 있을 것이다. 이는 함수의 다형성이 동작되는 것을 의미하며 R의 객체지향적인 성격을 보여주는 단적인 예이다.

```
> methods(predict)

##  [1] predict.ar*             predict.Arima*           predict.arima0*          predict.BinaryTree*
##  [5] predict.bs*             predict.bSpline*         predict.coxph*           predict.coxph.penal*
##  [9] predict.glinearModel*   predict.glm              predict.glmmPQL*         predict.goodfit*
## [13] predict.HoltWinters*    predict.lda*             predict.linearModel*     predict.lm
## [17] predict.loess*          predict.lqs*             predict.mca*             predict.mlm
## [21] predict.mob*            predict.nbSpline*        predict.nls*             predict.npolySpline*
## [25] predict.ns*             predict.pbSpline*        predict.polr*            predict.poly
## [29] predict.polySpline*     predict.ppolySpline*     predict.ppr*             predict.prcomp*
## [33] predict.princomp*       predict.pspline*         predict.qda*             predict.RandomForest*
## [37] predict.rlm*            predict.smooth.spline*   predict.smooth.spline.fit* predict.StructTS*
## [41] predict.survreg*        predict.survreg.penal*
##
##    Non-visible functions are asterisked
```

따라서 특정 'predict()' 함수의 도움말을 보고자 한다면 아래와 같은 방식으로 정확한 함수명을 입력해야 된다.

```
> ?predict.lm
```

R 기반의 객체지향 프로그래밍의 경우 패키지를 개발할 때를 제외하고는 거의 사용하지 않으며, 사용자를 위한 영역이라기 보다는 개발자를 위한 영역의 것이므로 이 책에서는 다루지 않겠으나 관심이 있으신 분들은 관련 문서를 찾아보길 바란다[11].

파일 읽고 쓰기

R에서 데이터를 읽어 오기 위해서 많은 인터페이스를 제공하고 있다. 일단 'read.*' 계열의 함수를 이용해 데이터를 읽어 오는 방법중에서는 'read.table' 함수가 대표적이며 이를 기반으로 'read.csv', 'read.csv2', 'read.delim' 과 같은 함수들이 있다.
이런 함수들의 첫 부분에는 'file' 인자가 들어있는데, 이곳에서는 'file()', 'url()', 'gzfile()' 과 같은 'connection' 을 생성할 수 있는 함수들이 올 수 있으며 직관적으로 'file' 인자에 파일 경로나 URL 텍스트를 적어도 인식하며 잘 동작한다.
'write.*' 계열의 함수는 'read.*' 계열의 함수의 역할과 정 반대의 역할을 하는 함수이다.

[11] `r-project.org`에는 상당한 양의 공개된 문서를 제공하고 있다. 특히나 `http://cran.r-project.org/other-docs.html`에 참고할 문서들이 많으니 관심 있는 독자분들은 읽어 보길 바란다.

```
> write.table(iris, file="book_resources/data/iris.table",row.names=FALSE)
> iris.table <- read.table("book_resources/data/iris.table")
```

만일 한글이 포함된 데이터 파일이라면 적절한 인코딩을 적어줄 필요가 있다. 가장 안전한 방법은 'UTF-8' 인코딩으로 저장을 하고 이를 읽어 들일때 'UTF-8'로 읽어들이는 것이다. 이미 많은 OS나 플랫폼에서 'UTF-8'을 기본 텍스트 인코딩으로 사용하고 있기 때문이다.
R의 데이터 뿐만 아니라 객체까지 저장하는 방법이 있는데 'save'와 'load'이다. 여러분들이 아마도 '*.Rda'파일이나 '*.RData'와 같은 파일을 본 적이 있다면 이는 R 객체를 저장하고 있는 바이너리 파일이라 생각하면 된다.

```
> save(iris, file="book_resources/data/iris.RData")
> load("book_resources/data/iris.RData")
```

필자의 경우 텍스트로 저장하기에는 너무 큰 객체일 경우 'compress="bzip2"'과 같은 압축 옵션을 추가적으로 적용해 'save()'함수를 실행켜 저장 용량을 크게 줄이는 방법을 사용하며, 이 기법은 꽤 유용하다. 그리고 작업공간 전체를 저장해 버리기 위해서는 'save.image()'함수를 사용하면 된다.
위 함수를 사용하는데 한가지 유의 사항이 있는데, 만일 데이터 내에 한글과 같은 2바이트 문자열이 포함되었을 경우 이 기종 OS 간에 다른 동작으로 인해 에러가 날 수 있다. 물론 한글만 들어 있지 않다면 이기종 OS간에 아무 무리 없이 상호 호환이 되나, 한글의 경우 해당 함수에서 인코딩 옵션을 제공하지 않기 때문에 각 OS 버전의 R에서 나름대로 한글 문자열을 현 실행 환경에서 유추 가능한 문자열 인코딩으로 디코딩을 시도하기 때문이다. 이 부분은 추후 보완이 될 거라 예상하나 그 전까지는 주의를 하기 바란다.

제어문

if(조건) 참일때 수행 else 거짓일 때 수행

if문은 위와 같은 형태를 띄고 있으며 단순히 'else'가 없이 사용되기도 한다. 많은 경우 'if' 문에 벡터형이 올 수 있는데, 이때에는 첫번째 원소만 평가된 결과가 리턴된다.

```
> x <- c(1, 10)
> y <- c(-3, 6, 7, -1, -2, 100)
> if( x < y) x else y

## Warning: length > 1 이라는 조건이 있고, 첫번째 요소만이 사용될 것입니다

## [1]  -3   6   7  -1  -2 100
```

'Warning'에 아주 자세한 설명이 나오는데 역시 첫번째 원소만으로 평가해서 값을 리턴하는 것을 의미하고 있다.
만일 벡터형으로 입력받아 연산을 하고 난 이후 벡터형으로 결과를 리턴받기를 원한다면 'ifelse'를 사용해야 한다.

```
> ifelse(x < y, x , y)

## [1] -3  6  1 -1 -2 10
```

아래와 같이 조건에 맞게 결과물을 반환해야 될 부분이 있다. 물론 아래와 같이 'if-else'문을 이용해서 할 수 있겠으나, 'switch'문을 활용하는게 더 직관적이며, 간략한 코드를 만들어 낸다.

```
> z <- 'R'
> language <- if(z == 'R'){
+   "R언어"
+ }else if(z == "C"){
+   "C/C++"
+ }else if(z == "PY"){
+   "Python"
+ }else{
+   "Etc"
+ }
> language

## [1] "R언어"

>
> language <- switch(z,
+                  R ="R언어",
+                  C ="C/C++",
+                  PY="Python",
+                  "Etc")
```

사실 같은 기능을 하는데 조금 간략한 코드를 만들어내는게 어떤 의미가 있을까 하고 생각하는 분들이 있을거 같은데, 코드라는것은 작성되는 순간부터 이미 다른 사람들에게 읽혀질 가능성을 가지고 있다. 따라서 적절한 설명과 함께 최대한 직관적이고 간단하게 코드를 만드는 습관을 가지는게 중요하다. 특히나 R의 경우 데이터를 접근하고 변환하며는 코드가 잘못 코딩하게 되면 한 라인이 굉장히 길어지게 되는 상황이 벌어지는데 이런 경우 코드가 유지보수하기 힘들어 지게 된다. 따라서 이럴 경우에는 어떻게 하면 소스코드를 간결하고 명확하게 만들 수 있는지 한번정도 고민하고 다른 표현방법을 생각해 볼 수 있어야 된다.
R에는 세 가지 반복문 구문이 존재한다. 그 첫번째는 'repeat'이다.

```
> a <- 4
> repeat{if(a > 10) break else {print(a); a <- a + 1; } }

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

```
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

만일 'break'가 없다면 무한 반복을 하게 될 것이다.

while (반복 조건) 표현

'while'문은 위와 같은 형태를 띄고 있다.

```
> a <- 4
> while(a <= 10){ print(a); a <- a + 1;}

## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

이전 코드와 같은 결과를 내는 코드이며 좀더 간결한 표현을 하고 있다. 'while'문은 다른 언어의 그것과 다르지 않은것을 볼 수 있다.
'for'문은 'while'과 'repeat'와는 좀더 다양한 반복을 가능케 한다.

```
> for(i in seq(1,10)){ print(i) }

## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10

> #홀수만
> for(i in seq(from=1,to=10,by=2)){ print(i) }

## [1] 1
## [1] 3
## [1] 5
## [1] 7
## [1] 9
```

물론 ‘break’ 문도 사용가능하며 다른 언어에서 존재하는 ‘continue’와 같은 역할을 하는 ‘next’라는 구문도 사용가능하다.

Chapter 3

Data munging with R

3.1 들어가며

2회에서는 R의 문법에 대한 개략적인 소개를 했으며, 아마도 R이라는 언어가 어떤 성격의 언어인지 느낌은 가지고 있을 거란 생각을 해본다. 사실 "시각화를 하자는데, 왜 이리 해야될 것들이 많아?" 하시는 분들이 있을 거란 생각을 해보는데, 실제 이런 작업이 필요하다. 왜냐면 시각화 패키지들은 그들 나름대로 입력 데이터 형식과 문법이 있다. 따라서 그것에 맞추어 데이터를 제공해 줘야 사용자가 원하는 시각화가 가능한 것이다.
금번 장에서 설명할 부분은 R로 하는 데이터 멍잉(data munging)이다. 사실 멍잉(munging)이라는 단어는 전처리, 파싱, 필터링과 같이 데이터를 이리저리 핸들링하는 행위를 의미하는 단어이다. 그런데 사전에 없다고 원고를 쓰는 이 시간에도 워드프로세서에서 빨간줄을 그어 놓는데, 사실 이 단어는 컴퓨터로 데이터를 처리하는 사람들 사이에서 많이 쓰이는 단어이고 아직 따끈따끈한 신조어이다.
R은 데이터 분석과 시각화에 강한 언어이다. 게다가 언어 자체의 데이터 멍잉 능력도 출중하지만, 여러 패키지의 도움으로 원본(raw) 데이터를 여러 가지 분석가가 원하는 형태로 아주 쉽게 변형이 가능해 쉽게 시각화를 위한 데이터를 만들거나 모델링을 위한 기반 데이터로 만들기가 굉장히 쉽다.
이번 회에서는 기본적인 R의 데이터 멍잉 기능을 먼저 살펴보고, 동일한 작업을 좀더 직관적이며 빠르게 수행할 수 있는 여러 패키지를 살펴보고자 한다. 이를 위해 'data.tabl','sqld','ply' 패키지를 살펴볼 예정이다. 'data.table'은 R에 기본적으로 탑재되어 있는 'data.frame'을 상속한 클래스로서 'data.frame'의 거의 모든 연산을 수행하며, 추가적으로 좀더 간편한 인터페이싱을 제공하고 있는 패키지이다. 필자는 이 패키지를 사용하면서 'data.frame'을 핸들링할 때보다 많은 코드를 줄였던 경험이 있다.
사실 'data.frame' 데이터형은 RDBMS의 table과 성격이 유사하다. 따라서 이를 SQL로 인터페

그림. 3.1: 가장 많이 사용되는 50가지 패키지들에 대한 시각화

이싱 하려는 노력이 있었는데, 'sqldf'가 바로 그런 역할을 한다. 몇몇 연산에서는 sqldf가 타의 추종을 불허할 만큼 빠른 경우도 있는데, 이 부분에 대해서는 뒤에서 알아보도록 하겠다.
'plyr' 패키지는 필자가 앞의 세 패키지 중에서 가장 먼저 사용한 패키지이며, 이를 만든 Hadley 교수가 다양한 유명 패키지에서 이를 사용하면서 더욱 유명해지게 되었다. 얼마전에 어느 블로거가 가장 많이 사용되는 50개 패키지를 소개한 워드클라우드가 있었는데, 아래와 같다[1].

우리가 다음에 배울 'ggplot2'이 가장 인기가 많은 패키지이며, 그 뒤를 따르는 것들이 'data.table'과 'plyr' 패키지라는것을 잘 알 수 있을 것이다. 'sqldf'는 성능에 비해서 아직 나온지 얼마 되지 않아 사용자가 많지 않은 것으로 생되는데 이 글을 쓰는 현재로서는 많은 DB관련 엔지니어들이 이 패키지를 즐겨쓰는 것을 쉽게 알 수 있었다.

3.2 R BASE 집계 함수 소개

SQL의 'select' 류의 함수를 R의 기본 데이터 연산과 비교해서 간단하게 설명을 하도록 하겠다. 아마도 통계학과나 컴퓨터공학과 혹은 개발자 경력을 가진 분들 이라면 SQL이라는게 어떤것을 의미하고 실제 사용도 해봤을 거란 생각을 해본다. 실제 통계학과나 컴퓨터공학과의 경우 해당 과목이 커리큘럼에 있으며, 개발 경력이 있다면 하물며 안드로이드 앱 개발이라고 하더라도 경량 DB가 포함되어 있기 때문에 SQL을 사용해 봤을 거란 예상을 해본다. 그리고 이전 장에서 'data.frame'이 RDBMS의 table과 유사한 성격을 가지고 있다는 것도 기억을 하고 있을 거란 생각을 해본다. 따라서 SQL을 기반으로 'data.frame'의 기본 연산에 대해서 좀더 살펴보도록 하겠다. 결론적으로 말씀 드리자면 'data.frame'을 핸들링 할 수 있는 방법은 정말

[1]출처 : `http://r-de-jeu.blogspot.kr/2012/04/50-most-used-r-packages.html`

많다. 같은 결과를 내는 다양한 방법이 존재하니 이 곳에 언급을 하지 않은 부분이 있을 수 있음에 대해서 이해 부탁 드린다.
기본적으로 iris데이터는 R 셸을 띄운 후 바로 사용이 가능한 데이터셋 이므로 이 데이터를 기반으로 설명을 해보겠다.

```
> #요약 통계
> summary(iris)

##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
##  Min.   :4.30   Min.   :2.00   Min.   :1.00   Min.   :0.1   setosa    :50
##  1st Qu.:5.10   1st Qu.:2.80   1st Qu.:1.60   1st Qu.:0.3   versicolor:50
##  Median :5.80   Median :3.00   Median :4.35   Median :1.3   virginica :50
##  Mean   :5.84   Mean   :3.06   Mean   :3.76   Mean   :1.2
##  3rd Qu.:6.40   3rd Qu.:3.30   3rd Qu.:5.10   3rd Qu.:1.8
##  Max.   :7.90   Max.   :4.40   Max.   :6.90   Max.   :2.5

>
> # 첫번째 행
> iris[1,]

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa

>
> #1~10row 까지 출력
> iris[1:10,]

##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1         3.5          1.4         0.2  setosa
## 2           4.9         3.0          1.4         0.2  setosa
## 3           4.7         3.2          1.3         0.2  setosa
## 4           4.6         3.1          1.5         0.2  setosa
## 5           5.0         3.6          1.4         0.2  setosa
## 6           5.4         3.9          1.7         0.4  setosa
## 7           4.6         3.4          1.4         0.3  setosa
## 8           5.0         3.4          1.5         0.2  setosa
## 9           4.4         2.9          1.4         0.2  setosa
## 10          4.9         3.1          1.5         0.1  setosa

> iris[c(1:10),]

##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1         3.5          1.4         0.2  setosa
## 2           4.9         3.0          1.4         0.2  setosa
## 3           4.7         3.2          1.3         0.2  setosa
## 4           4.6         3.1          1.5         0.2  setosa
## 5           5.0         3.6          1.4         0.2  setosa
## 6           5.4         3.9          1.7         0.4  setosa
```

```
## 7          4.6         3.4          1.4         0.3  setosa
## 8          5.0         3.4          1.5         0.2  setosa
## 9          4.4         2.9          1.4         0.2  setosa
## 10         4.9         3.1          1.5         0.1  setosa

>
> #꽃잎 길이가 2 이상인 행 출력
> iris[iris$Petal.Width > 2,]

##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 101          6.3         3.3          6.0         2.5 virginica
## 103          7.1         3.0          5.9         2.1 virginica
## 105          6.5         3.0          5.8         2.2 virginica
## 106          7.6         3.0          6.6         2.1 virginica
## 110          7.2         3.6          6.1         2.5 virginica
## 113          6.8         3.0          5.5         2.1 virginica
## 115          5.8         2.8          5.1         2.4 virginica
## 116          6.4         3.2          5.3         2.3 virginica
## 118          7.7         3.8          6.7         2.2 virginica
## 119          7.7         2.6          6.9         2.3 virginica
## 121          6.9         3.2          5.7         2.3 virginica
## 125          6.7         3.3          5.7         2.1 virginica
## 129          6.4         2.8          5.6         2.1 virginica
## 133          6.4         2.8          5.6         2.2 virginica
## 136          7.7         3.0          6.1         2.3 virginica
## 137          6.3         3.4          5.6         2.4 virginica
## 140          6.9         3.1          5.4         2.1 virginica
## 141          6.7         3.1          5.6         2.4 virginica
## 142          6.9         3.1          5.1         2.3 virginica
## 144          6.8         3.2          5.9         2.3 virginica
## 145          6.7         3.3          5.7         2.5 virginica
## 146          6.7         3.0          5.2         2.3 virginica
## 149          6.2         3.4          5.4         2.3 virginica

>
> #Petal.Width가 2보다 큰 행에서 Sepal.Length와 Sepal.Width 열만 출력
> iris[iris$Petal.Width > 2, c("Sepal.Length", "Sepal.Width")]      # ----------------------(1)

##     Sepal.Length Sepal.Width
## 101          6.3         3.3
## 103          7.1         3.0
## 105          6.5         3.0
## 106          7.6         3.0
## 110          7.2         3.6
## 113          6.8         3.0
## 115          5.8         2.8
## 116          6.4         3.2
## 118          7.7         3.8
```

```
## 119          7.7         2.6
## 121          6.9         3.2
## 125          6.7         3.3
## 129          6.4         2.8
## 133          6.4         2.8
## 136          7.7         3.0
## 137          6.3         3.4
## 140          6.9         3.1
## 141          6.7         3.1
## 142          6.9         3.1
## 144          6.8         3.2
## 145          6.7         3.3
## 146          6.7         3.0
## 149          6.2         3.4

> # Sepal.Length와 Sepal.Width 열은 첫번째, 두번째 열이므로 위 명령어와 같은 결과를 내준다.
> iris[1:10,1:2]

##    Sepal.Length Sepal.Width
## 1           5.1         3.5
## 2           4.9         3.0
## 3           4.7         3.2
## 4           4.6         3.1
## 5           5.0         3.6
## 6           5.4         3.9
## 7           4.6         3.4
## 8           5.0         3.4
## 9           4.4         2.9
## 10          4.9         3.1

>
> #종종 사용하는 테크닉
> iris[which(iris$Species == "setosa"),"Species"]

##  [1] setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa
## [17] setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa
## [33] setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa
## [49] setosa setosa
## Levels: setosa versicolor virginica
```

'data.frame'의 인덱스 연산은 대부분 아래와 같은 사용방식을 따른다.

DF[where, select]

따라서 (1) 코드는 SQL로 변환한다면 아래와 같다.[2]

select Sepal.Length, Sepal.Width from iris where Petal.Width > 2;

[2] ':' 문자는 SQL에서는 R과 다른 의미로 쓰이니 주의

더 추가하면 'DF[value]' 형식의 연산을 할 수 있기도 하다. 이 부분은 좀 헷갈릴 수 있으니 주의 바란다.

```
> a <- iris[1]
> b <- iris[,1]
> identical(a,b) #FALSE   ---- (1)

## [1] FALSE

> b <- iris[,1,drop=F]  # ---- (2)
> identical(a,b) #TRUE

## [1] TRUE

>
> a <- iris[1:2]
> b <- iris[,1:2]
> identical(a,b) #TRUE

## [1] TRUE

>
> a <- iris["Species"]
> b <- iris[,"Species",drop=F]
> identical(a,b) #TRUE

## [1] TRUE
```

'identical' 함수는 입력된 두 객체가 정확하게 같은지 비교하는 연산이다.
(1) 부분에서 FALSE가 나오는 이유는 두 연산에서 나오는 데이터 내용은 같지만 하나는 'data.frame'을 리턴하고 나머지 하나는 벡터를 리턴하기 때문이다. 이 부분은 초보자분들이 쉽게 틀릴 수 있는 부분이니 유념하고 넘어가야 한다. 따라서 'data.frame' 형식으로 무조건 리턴하게 만들기 위해 'drop=F(ALSE)' 옵션을 주니 비로서 같은 데이터를 리턴했다는 메시지를 볼 수 있다. 이 옵션의 경우는 어떤 열이 선택될지 모르는 상황에서 연산으로 리턴되는 데이터타입을 보장하기 위해서 주로 사용된다.
SQL의 많은 연산중에 하나는 특정 기준값에 대해서 집계를 하는 류의 질의어가 상당히 많이 사용되고 있다. 예를 들어 'group by' 같은 연산이 그 예이다. 하지만 아쉽게도 이와 같은 고수준의 데이터 연산을 'data.frame' 에서는 지원하지 않는다. 이를 기본 base 패키지만 가지고 수행해보고자 한다면 'tapply', 'aggregate', 'by' 류의 집계함수를 사용해야 한다.

3.3 tapply, aggregate, by 함수

먼저 iris데이터는 너무 재미 없고 심심하니 우리 생활과 밀접한 정보를 담고 있는 생필품 가격 데이터를 사용해 보고자 한다. 이 데이터의 출처는 `http://data.seoul.go.kr` 이며 이곳에 가입해서 사용해도 좋지만 실습 편의성을 위해 데이터를 dropbox[3]에 올려두고 바로 읽어 들이는 방식으로 코드를 소개 하겠다. 물론 위 링크에 가서 CSV파일로 저장한 뒤 그 데이터를 사용해도 무방하다. 데이터는 아래 순서와 같은 필드(열)로 구성되어 있다

영문명	한글명
P_SEQ	일련번호
M_SEQ	시장/마트번호
M_NAME	시장/마트이름
A_SEQ	품목번호
A_NAME	품목이름
A_UNIT	실판매규격
A_PRICE	가격
P_YEAR_MONTH	월-년도
ADD_COL	비고
M_TYPE_CODE	시장유형구분코드
M_TYPE_NAME	시장유형구분이름
M_GU_CODE	자치구 코드
M_GU_NAME	자치구 이름

Table 3.1: 생필품 가격 데이터 필드명 매핑

```
> head(market_price)

##    P_SEQ M_SEQ      M_NAME A_SEQ         A_NAME A_UNIT A_PRICE P_YEAR_MONTH ADD_COL M_TYPE_CODE M_TYPE_NAME
## 1 319572   199 서울중앙시장   305 사과(부사, 300g)    1개    2500       Jul-12  국내산           1    전통시장
## 2 319573   199 서울중앙시장   306   배(신고, 600g)    1개    3000       Jul-12  국내산           1    전통시장
## 3 319574   199 서울중앙시장   307    배추(2.5~3kg)  1포기    5000       Jul-12  국내산           1    전통시장
## 4 319575   199 서울중앙시장   308          무(1kg)    1개    1500       Jul-12  국내산           1    전통시장
## 5 319576   199 서울중앙시장   309    양파(1.5kg망)    1망    2000       Jul-12  국내산           1    전통시장
## 6 319577   199 서울중앙시장    23             상추   100g    1200       Jul-12  국내산           1    전통시장
##   M_GU_CODE M_GU_NAME
## 1    140000      중구
## 2    140000      중구
```

[3] `https://www.dropbox.com/`

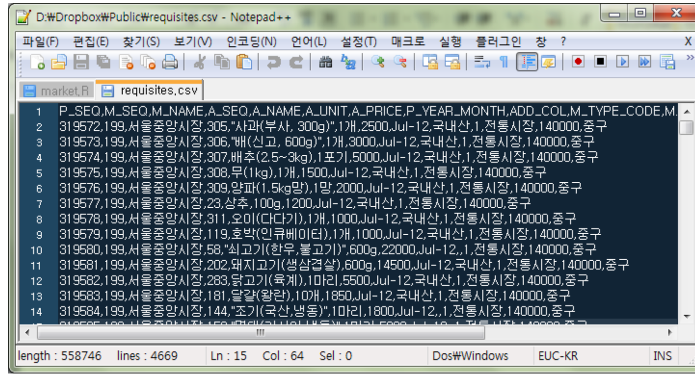

그림. 3.2: Notepad++에서 인코딩 확인

```
## 3    140000      중구
## 4    140000      중구
## 5    140000      중구
## 6    140000      중구

>
> nrow(market_price)

## [1] 4667

>
> summary(market_price)

##      P_SEQ           M_SEQ                M_NAME        A_SEQ                   A_NAME          A_UNIT
## Min.   :299094   Min.   :  1   2001아울렛불광점: 48   Min.   : 13   오이(다다기)        : 264   1개       :1163
## 1st Qu.:302232   1st Qu.: 49   경동시장         : 48   1st Qu.:134   닭고기(육계)        : 254   1마리     : 391
## Median :313531   Median : 94   고척근린시장     : 48   Median :266   쇠고기(한우,불고기): 242   600g      : 386
## Mean   :312873   Mean   :103   공릉도깨비시장   : 48   Mean   :220   고등어(생물,국산)  : 237   10개      : 269
## 3rd Qu.:322914   3rd Qu.:145   광장시장         : 48   3rd Qu.:307   달걀(왕란)          : 235   1포기     : 256
## Max.   :324241   Max.   :222   구로시장         : 48   Max.   :317   배(신고, 600g)      : 228   1마리(30cm): 135
##                               (Other)          :4379                 (Other)             :3207   (Other)   :2067
##     A_PRICE       P_YEAR_MONTH     ADD_COL       M_TYPE_CODE     M_TYPE_NAME     M_GU_CODE         M_GU_NAME
## Min.   :    0   Jul-12:1567         : 689   Min.   :1.00   대형마트:2383   Min.   :110000   관악구  : 240
## 1st Qu.: 1400   Jun-12:1533   국내산 : 257   1st Qu.:1.00   전통시장:2284   1st Qu.:260000   영등포구: 240
## Median : 2490   May-12:1567   국산   :  96   Median :2.00                   Median :410000   강동구  : 192
## Mean   : 4274                 제주   :  65   Mean   :1.51                   Mean   :424495   강북구  : 192
## 3rd Qu.: 4300                 무안   :  59   3rd Qu.:2.00                   3rd Qu.:560000   강서구  : 192
## Max.   :39000                 부산   :  43   Max.   :2.00                   Max.   :740000   광진구  : 192
##                               (Other):3458                                                   (Other) :3419
```

'read.csv' 명령어는 웹에 공개된 파일이든 로컬에 있는 파일이든 읽어와서 텍스트 인코딩에 맞게 읽어들여 'data.frame' 형식으로 리턴한다. 'fileEncoding' 옵션은 외부 파일을 읽어들일

때 항상 명시해 주는 습관을 들이는 게 좋다. 왜냐면 한글 표현에 있어서 문제가 생길 수 있기 때문이다. 그러나 인코딩을 잘 모르겠다는 독자는 윈도우나 맥에서 에디터를 열게 되면 문서 정보를 볼 수 있는데 그 부분을 확인하면 된다. 대부분의 국내 R 유저는 윈도우 유저니 윈도우일 때 확인 방법을 간단하게 알려드리면 프리웨어인 'Notepad++'[4] 에디터를 열고 그림3.2에서와 같은 프로그램 상에서 인코딩 정보를 확인하면 된다. 한글이 에디터에서 잘 보이고 트레이에 인코딩이 'ANSI', 'EUC-KR', 'Windows 949' 라고 표현되어 있으면 코드와 같이 'EUC-KR' 입력하면 되고 'UTF-8' 로 표시되어 있으면 fileEncoding='UTF-8' 로 하면 된다.
R에서 한글 데이터를 분석하기 위해서는 여러 기반 지식이 필요한데, 그중에 하나가 인코딩에 대한 이해이다. 이 부분에 대한 좀더 상세한 설명을 보고 싶으면 필자가 올려둔 동영상[5]을 확인하기 바란다.
인코딩 설명에서 서두가 길어졌는데, `market_price`변수에 'data.frame' 이 저장된다. 그리고 일상적으로 'head', 'nrow', 'summary' 와 같은 명령어를 입력해서 데이터가 어떻게 생겼는지 확인을 하는 과정을 통상적으로 하게 된다. 이때 확인하는 부분은 'data.frame' 데이터의 레코드가 자신이 알고 있는 레코드 개수와 맞는지 그리고 본인이 예상한 필드명과 그에 해당하는 데이터들이 적합한지 개략적으로 확인한다. 'summary' 명령어를 보면 대략적인 데이터 분포를 필드별로 확인이 가능하여 굉장히 유용하다.
그럼 이제 데이터로 돌아가자!
이 데이터는 전통시장과 대형마트에서 생필품 가격이 어떻게 되는지 직접 공무원 분들이 조사를 나가서 수집한 데이터이다. 물론 지역에 따라서 모두 구분이 되어 있고, 시장과 마트구분은 물론, 마트이름과 시장이름도 확인 가능하다.
여기서 여러분들이 가장 먼저 알고 싶은건 어떤 정보인가? 필자는 직접 장을 자주 보기 때문에 전통시장과 대형마트간의 가격차이가 있는지 확인을 해보고 싶었다. 물론 품목별로 평균을 뽑아서 적절한 데이터 포맷으로 보여주면 될 것이다.
R은 특정 데이터 집단에 함수를 적용하는 'tapply', 'by', 'aggregate' 와 같은 함수를 제공한다. 이들의 동작원리는 대략 그림3.3와 같다.

'split', 'apply', 'combine' 라는 전략을 사용하는 방식으로 특정 기준에 의해서 데이터를 그룹화 하고 개별적으로 처리해 마지막에 결합하는 효율적인 전략을 사용하는데, 이런 개념은 'Hadoop' 의 'map/reduce' 와 닮아 있다. 참고로 R에서도 'Map()', 'Reduce()' 라는 함수형 언어에서 가져온 함수가 포함되어 있다. 'tapply', 'by', 'aggregate' 함수들은 모두 위와 같은 전략을 따르며 입,출력 포맷에 따라 다른 이름을 가지고 있을 뿐이다.

'tapply' 를 가지고 대형마트, 전통시장에 따른 평균 물품 가격을 뽑아보는 명령어는 아래와

[4] `http://www.notepad-plus-plus.org/`
[5] `http://www.youtube.com/watch?v=araCWRa5Nxg`

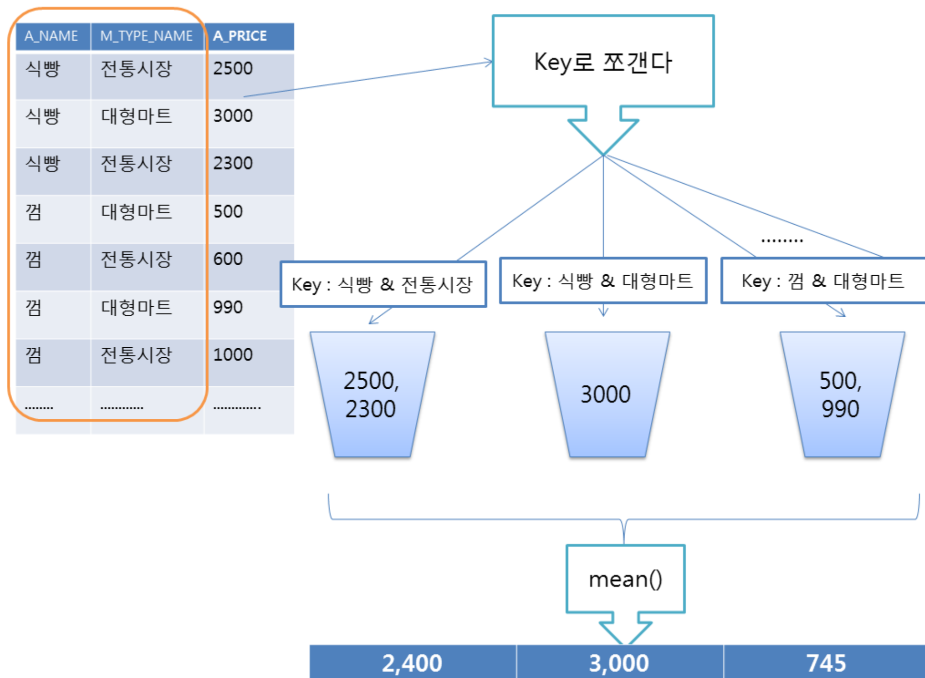

그림. 3.3: Split-Apply-Combine 프로세싱 패턴

같다.

```
> tapply(market_price$"A_PRICE", market_price[,c("M_TYPE_NAME","A_NAME")], mean)
```

함수의 원형은 `tapply(X, INDEX, FUN = NULL, ..., simplify = TRUE)` 형태와 같으며 X는 우리가 확인하고자 하는 가격이 되겠고, INDEX는 factor형으로 구성된 리스트를 입력한다. 물론 의문을 가질 수 있다. 왜냐면 위의 예제는 사실 'data.frame'이 인자로 들어갔기 때문이다. 이제야 언급하지만 'data.frame'은 리스트이기도 하다. 물론 리스트에 대해서는 자세한 설명을 하지 않았지만 'data.frame'의 연산방식에 리스트의 특징도 들어가 있는 것을 알 수 있을 것이다. 그리고 마지막 인자는 FUN에는 함수명을 입력한다. 이렇게 나오는 코드는 행이 M_TYPE_NAME이고, 열이 A_NAME으로 구성된 행렬데이터를 리턴한다. 물론 그 내부 값들은 모두 평균값이 계산된 결과이다.
'tapply'는 키가 하나이고 FUN으로 입력된 함수의 리턴값이 하나라면 벡터형의 데이터를 반환하고, 이것보다 복잡한 데이터가 리턴될 경우 행렬이나 리스트형태로 반환 된다.

```
> by(market_price$A_PRICE, market_price[,c("M_TYPE_NAME","A_NAME")], mean)
```

위의 명령어도 있다. 역시 비슷한 작업을 하나 작업 방식이 서브 'data.frame'을 키를 기준으로 분리한 뒤 각 'data.frame'에 'mean'함수를 적용해서 이들을 리스트로 엮어서 리턴한다.

```
> aggregate(market_price$A_PRICE, market_price[,c("M_TYPE_NAME","A_NAME")] , mean)
> aggregate( A_PRICE ~ M_GU_NAME + A_NAME, market_price, mean)
```

위의 'aggregate' 함수는 이전 함수들과 같은 연산을 하지만 리턴되는 데이터가 'data.frame'이다. 사실 대부분의 경우 'aggregate' 함수로 집계를 한 결과가 보기에 편하고 직관적이다. 아니면 'data.frame'이 데이터를 핸들링하고 보는데 익숙해서 그럴 수도 있다[6].

```
##   M_TYPE_NAME               A_NAME    x
## 1     대형마트               고등어 1000
## 2     대형마트   고등어(30cm,국산) 5335
## 3     전통시장   고등어(30cm,국산) 2667
## 4     대형마트   고등어(냉동,국산) 2825
## 5     전통시장   고등어(냉동,국산) 3525
## 6     대형마트 고등어(냉동,수입산) 1500
```

위의 결과는 'aggregate' 명령어의 결과로 나온 'data.frame'의 첫 부분이다. 그리고 'ggplot2'에서는 이런 작업을 사용자 모르게 내부적으로 수행해서 집계된 결과를 플로팅 해준다. 그리고 이를 해주는 패키지가 'plyr'이고, 역시 'ggplot2'의 개발자인 Hadley교수가 필요에 의해서 만들어졌던 것이다.

3.4 plyr 패키지

그럼 이제 'plyr' 패키지를 활용한 방법에 대해서 설명할 때가 된 듯 하다.
여담이지만 이 패키지의 개발자인 Hadley교수는 R커뮤니티 내에서도 굉장한 인기를 가지고 있다. 필자가 UseR! 2012 행사에 참석해 직접 이분이 발표하는 세션을 듣고자 했는데, 강의실 제일 뒤에서 서서 들어야 되었으며 강의실 사이에 난 통로까지 사람들이 빼곡히 들어차 그 엄청난 인기를 실감할 수 있었다. 대단한 내용이 아닌 발표임에도 그 정도 사람이 모일 정도였다. 그러나 가장 중요한 것은 교수님의 R패키지 개발 동기였다. 그분의 동기는 다른 사람이 자신이 겪은 시행착오를 겪지 않고 편하게 분석할 수 있게 하고 싶어서 패키지를 개발하게 되었다고 말씀하셨는데, 패키지들의 완성도만큼이나 훌륭한 동기를 가지고 계셨다. 아마도 그런 열정과 공공에 대한 재능기부가 그 인기를 만들었으리라 생각해본다.
'plyr' 패키지 이외에도 앞서 첨부한 워드클라우드에서 'reshape'라는 패키지를 볼 수 있을 것이다. 이 역시 데이터를 핸들링하고 변형하는 패키지이며, Hadley 교수가 만들었다. 물론 이 패키지를 이용해서도 유사한 데이터 멍잉 작업을 수행 할 수 있지만 성격은 약간 다르다. 'plyr'이 split, apply, combine의 전략으로 데이터를 그룹해서 집계 처리하는 것과는 다르게 집계하기보다는 데이터의 정보의 손실 없이 데이터의 형태를 바꾸는 작업이 주된 작업이다(물론 집계도 가능하지만 패키지의 목적은 집계가 아니다). 그래서 이름도 reshape이다. reshape에 대한

[6] 엑셀의 시트 혹은 RDBMS의 테이블이 이와 비슷한 성격인것 한몫하는거 같다.

설명은 여기서 다루지 않겠지만 관심 있는 독자 분들은 패키지의 목적 정도는 추후 파악해보는 시간을 갖는걸 추천한다[7].

'split-apply-combine' 전략은 많은 데이터 처리 업무에 대한 프레임웍을 제공해 줬다. 쪼개고 적용하는 부분을 'map' 이라하고 이를 다시 결합하는 부분을 'reduce' 라고 불러 이를 프레임웍으로 만든게 'Hadoop' 이라 생각하면 된다. 이는 대용량 데이터를 처리하는데 효과적이며 내부의 데이터가 어떤 것이든지 프레임웍이 적용하는 사용자로 하여금 간단하게 설정 가능하게한 유연성을 제공해 왔다. 그러나 한계도 있는데, split 된 이전 데이터 조각의 정보를 가져와 현재 조각에서 뭔가를 하고자 하는 연산은 이 프레임웍에 적합하지 않다. 물론 이 부분은 'map/reduce' 도 그렇다.

사실 이 패키지의 목적은 SQL의 'group by' 와 같은 집계를 편하게 하기 위함이며, 'Hadoop' 과 같이 분산처리를 위한 목적이 주된 목적은 아니다. 하지만 여러분의 머신이 멀티코어라면 조각난 데이터에서 함수를 적용하는 부분을 멀티코어분산처리가 가능하게 끔 하는 옵션을 제공하고 있다.

'plyr' 은 입,출력 데이터 포맷에 따라 아래 Table 3.2와 같이 쓰일 수 있는 함수를 나눠볼 수 있다.

	array	data.frame	list
array	aaply	adply	alply
data.frame	daply	ddply	dlply
list	laply	ldply	llply

Table 3.2: 입출력에 따른 함수 이름 매핑

이중에서 가장 많이 쓰이는 'ddply' 를 사용해 이전에 했던 집계 작업을 해보도록 하겠다.

```
ddply(.data, .variables, .fun = NULL, ...,
.progress = "none", .drop = TRUE, .parallel = FALSE)
```

data는 입력 data.frame이며, '.variables' 는 쪼개는 기준이 되는 값이며, 아래와 같은 형태중에 하나가 입력된다.

- 문자벡터 : `c("M_GU_NAME", "A_NAME")`
- 숫자벡터 : `c(5, 13)`
- 수식 (Formula) : `~ M_GU_NAME + A_NAME`

[7]Hadley 교수의 Tidy Data(Wickham, Submitted)라는 글에서 이 패키지의 중요 쓰임새가 많이 소개된다.

- 특별식 : .(M_GU_NAME, A_NAME)

네번째 특별식은 'plyr' 패키지에서만 쓰이는 방식이다.
fun은 함수명이 들어가며, '...'은 입력된 함수명에 추가되는 인자들이다. progress는 수행시간이 오래 걸릴경우를 대비해 진행상황을 표시해 주게끔 하는 옵션이다. 'drop=TRUE'는 현재 쪼개는 조건이 데이터에 있는 경우들만 계산해 출력하게 하는 옵션이다. 만일 이게 FALSE라면 모든 가능 조건에 대해서 출력을 하게 된다. '.parallel' 옵션은 머신의 멀티코어를 사용할 수 있게 해 주는 옵션이다. 멀티코어를 효과적으로 사용하기 위해서는 데이터 크기와 처리시간을 고려해서 사용해야 되는데 이 부분은 고성능 리눅스 서버를 사용한다는 가정하에 설명을 하도록 하겠다.

```
> # 싱글 프로세싱 하는 경우
> system.time({
>     a1 <- ddply(market_price, .(A_NAME, M_TYPE_NAME), summarize, mean_price = mean(A_PRICE))
> })
>
> # doMC를 사용한 멀티코어 프로세싱
> library(plyr)
> library(doMC)
> registerDoMC()
>
> system.time({
>     a2 <- ddply(market_price, .(A_NAME, M_TYPE_NAME), summarize, mean_price = mean(A_PRICE),
>         .parallel = TRUE)
> })
```

멀티코어 프로세싱이 모든 경우에 싱글보다 빠르지는 않다. 단, 'group by' 된 한 단위의 연산량이 많고 데이터가 클 경우 싱글보다 훨씬 빠르게 동작한다. 이런 효과를 보기 위해서는 고성능[8] 리눅스 분석 서버에 RStudio Server를 설치하고 일반적인 클라이언트에서 작업을 수행하는 경우이다.
'registerDoMC' 함수의 핵심적인 역할은 코어를 몇개로 정하는지인데 인자를 주지 않고 함수를 실행하면 자동으로 시스템 리소스에 맞게 정해준다. 하지만 만일 프로세싱이 끝날 기미가 보이지 않으며 cpu load가 미비하며, swap이 채워져 있다면 이 함수를 이용해서 코어수를 적게 조절해 줄 필요가 있다. 이런 이유 때문에 고성능 리눅스 서버를 사용하라고 필자가 언급을 했던 것이다.

```
> ddply(market_price, .(M_TYPE_NAME, A_NAME), summarise, avg=mean(A_PRICE))
```

위 코드는 우리가 목적으로 하는 결과를 뽑기 위한 코드이다. 사실 위에서 '.fun'에 해당하는 코드들이 바로 'summarise' 이하 부분이다. 'summarise' 함수는 그룹화 된 data.frame을 요

[8] 여기서 고성능이라 함은 이 글을 쓰는 현재 기준으로 메모리가 128GB이상이 되고 CPU는 쿼드코어 이상을 의미한다.

약하는데 쓰이는 함수이다. 따라서 'summarise'의 첫번째 인수에는 쪼개는 기준으로 쪼개진 'data.frame' 조각들이 들어가게 되며, avg라는 새로운 열로 각 쪼개진 데이터에서 A_PRICE 열의 평균값들을 넣게 된다. 대부분 'summarise'를 넣어서 사용하긴 하지만 간혹 'transform' 함수를 넣어 사용하기도 한다. 'transform'함수는 기본 base패키지에서 제공하는 함수로 기존의 'data.frame'에 새로운 열을 추가한다든지 혹은 기존의 열을 다른 데이터로 채워 넣을 때 사용한다. 'summarise'함수가 avg에 대한 하나의 열만 반환하지만, 'transform'은 열을 추가한 'data.frame'조각 전체를 반환하게 된다. 사실 설명만으로 두 함수의 구별점을 찾기는 힘든데, R을 학습하면서 가장 좋은 습관중에 하나는 특정 데이터를 가지고 함수가 실제 어떻게 동작하는지 확인하는 것이다. 위 두 함수도 어떻게 동작하는지 독자들이 꼭 한번 실습해보길 바란다.
마지막으로 연산 처리 속도가 가장 빠르다고 자타가 공인하고 있는 'data.table'의 코드를 살펴보도록 하겠다. 사실 'data.table'이 빠른 이유는 RDBMS에서 테이블에 인덱스를 거는 것과 같은 작업을 명시적으로 해주기 때문이다. 특정 키에 대해서 sequential search를 하는 'data.frame'보다 binary search를 하는 'data.table'이 빠른건 어찌 보면 당연한 이야기일 수 있겠다.
'data.table'의 경우 'data.frame'을 상속해서 대부분 유사한 사용방법을 공유하고 있다. 하지만 이를 굳이 'data.frame'을 통해 구현하지 않은 이유는 사용 편의성과 유연성을 제공하기 위해 기존의 'data.frame'과 혼용시 오류를 불러올 수 있는 부분들이 있기 때문이다. 따라서 'data.table'을 'data.frame'을 기반으로 해서 접근하는 것은 가장 쉬운 접근 방식이나, 반드시 자신이 의도한 바가 맞는지 메뉴얼을 확인해 사용을 해야 한다. 왜냐면 미묘하게 다르게 동작하는 부분들이 있기 때문이다.

3.5 data.table 패키지

'data.table'의 개략적인 문법은 아래와 같다.

```
> dt[i, j, mult={'first', 'last', 'all'},
>         nomatch={0, NA},
>         roll={FALSE, TRUE},
>         by='colname']
```

i,j에 들어가는 값은 'data.frame'의 그것과 크게 다르지 않다. i는 행을 선택하는 문법이며, j는 열을 선택하는 문법이 들어가게 된다. 그리고 나머지 옵션에서 mult는 'JOIN'연산시 키 매칭의 처음, 마지막, 모두를 리턴하는 옵션을 의미한다. nomatch의 경우 'OUTER JOIN'을 사용할 시 사용하게 되며 roll은 시계열 데이터에 대한 'rolling join'을 하게 될 때 가장 최근의 시간 데이터를 활용하게 끔 하는 옵션을 의미한다. by의 경우 'GROUP BY'연산을 할 때 사용하는 옵션으로 우리가 필요로 하는 옵션이다.

```
> library(data.table)
>
> market_price.dt <- data.table(market_price) # --- (1)
>
> market_price.dt[2,list(M_NAME)] # --- (2)

##          M_NAME
## 1: 서울중앙시장
```

우리가 지금까지 사용한 'data.frame' 객체인 market_price을 'data.table()' 함수를 이용해 'data.table'로 변환해주는 코드(1)와 변환된 객체에서 두번째 행의 M_NAME 열의 데이터를 가져오는 코드를 보여준다(2). 사실 (2)번 코드는 기존의 'data.frame'을 이용할 경우 아래의 코드로 가능하다.

```
> market_price[2,"M_NAME",drop=F]

##          M_NAME
## 2 서울중앙시장
```

이들 다르게 동작하는 코드에는 상당히 많은 내용이 함축되어 있다. 일단 drop=F를 사용한 이유에 대해서 설명하면 아래와 같다.
'data.frame'의 경우 한 열을 리턴할 경우 벡터로 리턴하고 두 개 이상의 열을 리턴할 경우 'data.frame'으로 리턴하는 상당히 특이한 방식을 사용한다. 이 때문에 함수를 작성하는데 어려움을 겪을 때가 종종 있다. 물론 'data.frame'으로 해결할 수 있는 방법도 있는데 바로 'drop=F'가 그런 역할을 한다.
이런 어려운 점을 해결하기 위해 'data.table'의 경우 리스트로 열 이름을 입력받게 하고 모든 경우에 'data.table'로 리턴하게 했다. 이는 'data.frame'을 오용하는 것을 원천에 방지하게끔 한다.

```
> market_price.dt[,list(avg = mean(A_PRICE)), by=list(M_TYPE_NAME, A_NAME)]

##      M_TYPE_NAME              A_NAME  avg
##   1:    전통시장    사과(부사, 300g) 2524
##   2:    전통시장      배(신고, 600g) 3099
##   3:    전통시장       배추(2.5~3kg) 2724
##   4:    전통시장             무(1kg) 1415
##   5:    전통시장       양파(1.5kg망) 2209
##  ---
## 126:    전통시장              오징어 1660
## 127:    대형마트 고등어(냉동,수입산) 1500
## 128:    대형마트              고등어 1000
## 129:    전통시장 고등어(생물,수입산) 1500
## 130:    대형마트 오징어(생물,수입산) 1680
```

M_TYPE_NAME과 A_NAME으로 'GROUP BY'된 각각의 'data.table'에 avg라는 열을 입력해 A_PRICE의 평균값을 넣는 연산을 의미한다. 물론 'by'에 들어가는 변수는 여러개가 될 수 있으며, 연산을 통해 만들어지는 avg같은 변수들도 여러개가 될 수 있다. 이는 기본 base 패키지에서 제공하는 집계함수에서는 보기 힘든 연산 방식이며, 'plyr'에서도 이와 같은 연산을 지원한다.
마지막으로 'sqldf'를 활용한 방법을 간단하게 살펴보겠다.

```
> library(sqldf)
> sqldf("select M_TYPE_NAME, A_NAME, avg(A_PRICE) as avg
+       from market_price group by M_TYPE_NAME,A_NAME limit 10")

##    M_TYPE_NAME                A_NAME    avg
## 1     대형마트                  고등어 1000.0
## 2     대형마트        고등어(30cm,국산) 5335.0
## 3     대형마트        고등어(냉동,국산) 2825.0
## 4     대형마트      고등어(냉동,수입산) 1500.0
## 5     대형마트        고등어(생물,국산) 4550.5
## 6     대형마트  냉동참조기(20cm,국산)  824.7
## 7     대형마트                    달걀 2472.5
## 8     대형마트              달걀(왕란) 2614.3
## 9     대형마트              달걀(특란) 2130.0
## 10    대형마트                  닭고기 6573.5
```

사실 이 패키지의 대부분을 차지하는 함수는 바로 'sqldf()'라는 함수이다. 아마도 독자분들이 SQL에 익숙하시다면 위 'select'문이 어떤것을 의미하는지 바로 아실거라 생각하고 이전에 설명한 여러 함수들과 동일한 결과를 의도한다는 것을 바로 눈치 채셨을 거라 생각한다. 바로 위의 함수 사용 예 처럼 'sqldf'는 'data.frame'을 흡사 데이터베이스의 테이블처럼 여기며 sql문으로 조작할 수 있는 인터페이스를 제공한다. 이는SAS의 SQL procedure와 흡사하며, 'sqldf()'안에서 대부분의 SQL쿼리가 수행 가능하고 'avg'같은 SQL 내장 함수도 호출 가능하다.
'plyr'이나 'data.table', 'sqldf'를 설명하면서 반드시 나오게 되는 내용은 퍼포먼스 차이이다. 이미 언급한 것처럼 알고리즘 관점에서 아래와 같은 특정 연산시 'data.table'이 더 빠르다는 것을 예상 할 수 있다.

```
> system.time(ddply(market_price, .(M_TYPE_NAME, A_NAME), summarise, avg=mean(A_PRICE)))

##    user  system elapsed
##    0.08    0.00    0.08

> system.time(market_price.dt[,list(avg = mean(A_PRICE)), by=list(M_TYPE_NAME, A_NAME)])

##    user  system elapsed
##       0       0       0
```

```
> res <- ddply(market_price, .(M_TYPE_NAME, A_NAME), summarise, avg=mean(A_PRICE))
> ggplot(res, aes(A_NAME, M_TYPE_NAME)) + geom_tile(aes(fill=avg)) +
+   theme(axis.text.x=theme_text(angle=90, hjust=1)) + scale_fill_gradient(low="green", high="red")
```

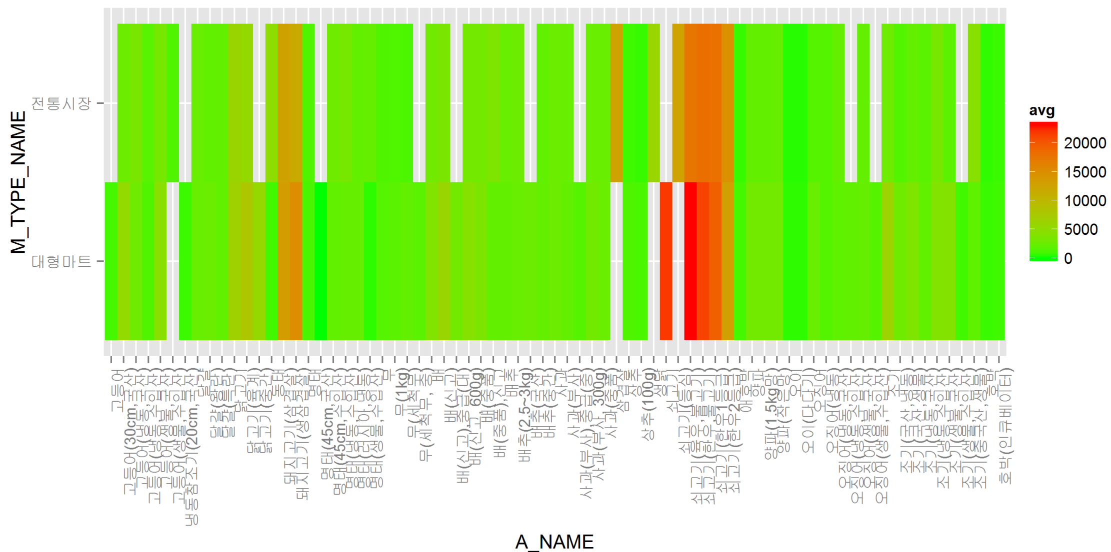

그림. 3.4: 품목에 대한 시장별 가격 분포

```
> system.time(sqldf("select M_TYPE_NAME, A_NAME, avg(A_PRICE) as avg from
+                   market_price group by M_TYPE_NAME,A_NAME"))

##    user  system elapsed
##    0.05    0.00    0.05
```

'system.time'은 인자로 주어진 표현식에 대해서 수행 소요 시간을 리턴하는 함수이며, 위의 결과를 보면 'data.table', 'sqldf', 'plyr'순으로 속도가 빠른 것을 알 수 있다. 물론 이 소요시간은 각자 코드를 돌리는 환경에 따라 다를 것이다.

3.6 왜 멍잉(munging)을 하느냐?

사실 왜 데이터를 이렇게 저렇게 조작하는게 시각화 작업에 필요한지 충분한 설명을 하지 못한게 사실이다. 실제 이 변환된 데이터를 기반으로 확인할 수 있는건, 전통시장, 대형마트 그리고 품목에 따른 가격 차이이다. 원 데이터로는 이 결과를 확인할 수 없다. 따라서 변환을 시도한 것이며, 이 변환된 결과가 아래 그림3.4과 같은 플로팅 결과로 나오게 된다.

그림 한 장으로 품목간의 대략적인 가격 차이와 마켓형태에 따른 가격 차이의 경향을 볼 수 있다. 따라서 위와 같은 플로팅 결과를 보기 위해 그에 따른 적절한 변환이 필요한 것이며, 바로

이런 이유 때문에 우리가 이번 장에서 데이터 멍잉을 배우는 것이다(물론 멍잉의 목적이 시각화를 위한 것만은 아니다). 물론 우리가 추후 배울 'ggplot2' 도 그룹 연산을 통해 데이터를 시각화해주는 옵션이 내부적으로 존재하지만, 그 옵션 자체가 유연성이 많지 않아 직접 이런 식으로 데이터를 변환해 줘야 하는 것이다.

Chapter 4

ggplot2를 이용한 R 시각화

드디어 핵심 주제에 다다랐다. R 기본 'graphics'를 건너뛰고 이렇게 'ggplot2'를 소개하는 이유는 다음과 같다. 'ggplot2'는 기본 R 그래픽스에서 제공하는 대부분의 작업을 아주 효과적으로 수행할 수 있어서 많은 책에서 시각화의 기본 패키지로 이미 'ggplot2'를 사용하고 있기 때문이다.
필자도 기본 R 그래픽을 주로 사용해 왔다. 기본 R 그래픽은 확장성을 보장하지만 기본적으로 미려한 그래픽 측면에서 아쉽고 설정도 복잡하다. 망설이고 있던 차에 한 책에서 'ggplot2'를 만나 본격적으로 사용하기 시작했다.
필자가 'ggplot2'를 실무에 적용하면서 느낀 장점들 중에서 가장 마음에 들었던 것은 기본 옵션으로 플로팅해도 아주 실용적인 색상 조합과 더불어 미려한 그래픽으로 정보를 시각화해 준다는 것이다. 이는 미적 감각이 부족한 필자에게 더없이 매력적이었다. 게다가 컬러도 눈에 잘 들어오도록 아주 세심하게 조합해 준다. 이를 흑백 프린터에 출력하더라도 범주값에 따른 색상 구분이 눈으로 가능해, 작은 부분 하나까지 꼼꼼하게 신경 썼다는 느낌을 주기에 충분했다. 특히 'Grammar of Graphics'의 개념은 흡사 객체지향(OOP) 프로그래밍 언어를 처음 배울 때의 느낌과 비슷했다. 데이터 객체, 그래픽 객체의 분리와 재사용에 초점을 둬 그 논리 정연함이 기본 R 패키지의 그것보다 훨씬 뛰어남을 느낄 수 있었다. 2005년부터 이를 개발하기 시작한 'Hadley Wickham'교수는 R 기본 그래픽 시스템의 장점을 가져와 발전시키는 데 또 한 가지 초점을 두었기 때문에 이런 탁월한 느낌을 줄 수 있었던 것 같다.

4.1 왜 ggplot2이 필요하나?

얼마 전 필자는, Hadley교수가 강연 중에 한 참석자로부터 '왜 사람들이 R 기본 그래픽에서 ggplot2로 바꿔야 하나요?'하는 질문을 받고 답하는 장면 영상을 보았다. 이에 대해 Hadley 교

수는 '기본 그래픽 시스템은 그림을 그리기 위해 좋은 툴이지만, ggplot2는 데이터를 이해하는 데 좋은 시각화 툴이기 때문'이라고 답했다. 진짜 이유는 'ggplot2'의 문법에 기인하는데, 기본 그래픽에서는 데이터 입력 포맷에 대한 함수 내 규정을 상세하게 하고 있어 한번 그려 놓기도 번거로울뿐더러 같은 데이터로 다른 형식의 그래프를 그릴 때 다른 입력 포맷에 대한 적응과 적용이 필요하다. 그러나 'ggplot2'는 다른 그래프라 하더라도 문법 내에서 간단한 코드 추가/삭제로 플로팅을 할 수 있다. 하나의 데이터를 기반으로 이런저런 그래프를 많이 그려봐야 될 현재 빅 데이터 시대에서 가장 필요한 기술 중에 하나가 아닐까 하는 생각을 해본다.
'ggplot2'를 설치하고 사용하기 위한 명령어는 아래와 같이 단 두줄이다.

```
> install.packages('gplot2')
> library(ggplot2)
```

개발 버전을 사용해볼 생각이라면 아래와 같은 명령어를 사용하면 된다.

```
> install.packages('devtools')
> library(devtools)
> #인스톨된 버전에 덮어쓰지 하지 않기 위해
> dev_mode()
> install_github('ggplot2')
```

ggplot2의 매뉴얼은 R help시스템에서 제공하는 것과 온라인으로 제공하는 것이 있는데, 가독성 측면에서 온라 메뉴얼[1]을 보는 것을 추천한다. 이 링크에서는 100여 개의 ggplot2 객체에 대한 소개와 함께 500여 개가 넘는 예제를 제공한다. 이 온라인 페이지는 늘 가까히 두고 익힐 필요가 있다. 그리고 필자가 ggplot2를 공부하면서 참고한 책은 'ggplot2: Elegant Graphics for Data Analysis'(Wickham, 2009)이다. ggplot2에 대한 유일한 책으로 Hadley 교수가 친근한 단어로 쉽게 소개하고 있다. 다양한 시각화 기술을 포함하고 있으므로 'ggplot2'를 심도 있게 공부하고 싶은 독자께서는 반드시 봐야 될 책이라 할 수 있다. 그리고 이 글 역시 위 두 참고문헌을 기반으로 작성했음을 밝혀둔다.
일단 동기 부여는 이쯤에서 마치고, 실제 이 패키지의 근간이 된 'Grammar of Graphics'에 대해 알아보자. 'Grammar of Graphics'는 데이터를 각 기하 객체(geometric object)의 미적 속성(aesthetic attributes)에 매핑하는 방법을 제공한다. 이를 통해 통계적인 시각화를 가능하게 하는 효과적인 방법을 제안한다. 이 와중에 통계적인 데이터 변환이 필요하다면 그 변환까지 수행해 준다. 국소(faceting) 시각화 기법을 사용하면 각 데이터의 부분 데이터만 사용해 여러 개의 그래프를 한꺼번에 그려주기도 한다.
먼저 문법(Grammar of Graphics) 설명 차원에서 데이터 하나를 소개한다.
ggplot2를 로딩해 그 안에 포함된 데이터를 사용해 볼 수 있다. diamonds 데이터도 역시 패키지와 함께 배포되는 데이터이며 이에 대한 필드 설명은 아래 테이블4.1와 같으며, 데이터에는

[1] http://docs.ggplot2.org/

5만 4000여 개의 다이아몬드에 대한 가격을 포함한 10개의 속성 정보가 포함돼 있다. 아마도 다이아몬드의 가치에 대해 궁금한 독자라면 시각화에 관심이 더 갈 것이다.

필드명	설명
price	가격 ($326 - $18,823)
carat	무게 (0.2, 5.01)
cut	컷팅의 가치 (Fair, Good, Very Good, Premium, Ideal)
colour	다이아몬드 색상(J(가장 나쁜)에서 D(가장 좋은)까지)
clarity	깨끗함 (I1 (가장 나쁜), SI1, SI2, VS1, VS2, VVS1, VVS2, IF (가장 좋은))
x	길이 (0, 10.74mm)
y	너비 (0, 58.9mm)
z	깊이 (0, 31.8mm)
depth	깊이 비율 = z / mean(x, y)
table	가장 넓은 부분의 너비 대비 다이아몬드 꼭대기의 너비(43- 95)

Table 4.1: diamonds 데이터셋 설명

'ggplot2'에서는 qplot라는 함수를 제공한다. 이는 R에서 기본적으로 제공하는 plot()과 유사한 인터페이스를 제공하는데 목적을 둔 함수다. 이를 사용하는 예는 다음과 같다.

```
> qplot(diamonds$carat, diamonds$price)
> qplot(carat, price, data = diamonds)
> qplot(carat, price, data = diamonds, geom="point" ,colour=clarity) # -- (1)
> qplot(carat, price, data = diamonds, geom=c("point", "smooth"), method=lm)
> qplot(carat, data = diamonds,geom="histogram")
```

'qplot()' 함수의 형태는 R base의 'plot()'과 유사하게도 첫 번째 인자에 x축의 변수명이 오며, 이어서 y축의 변수명이 따라온다. 그리고 data라는 인자로 'data.frame' 형태의 데이터의 객체명을 입력받는 형식이다. 뒤 이어 오는 인자들은 ggplot2에서만 해당되는 특수 인자들이 오게 된다.
예를 들어 앞 코드 (1)을 실행한 결과는 다음과 같다.
이 그래프를 코드와 매핑해 보면 아주 쉽게 그 의미를 파악할 수 있다. 이 그래프는 x축은 caret를, y축은 price를 의미한다. 그래프의 종류는 각 점마다 point를 찍는 scatter plot을 그려준다. 이와 동시에 다이아몬드의 선명도(clarity)에 따른 컬러도 매핑을 시켜줬다.

앞으론 'qplot()'에 대한 설명은 더 이상 하지 않을 생각이다. 여기엔 이유가 있다. 일단 'ggplot2'를 다양하게 활용하기 위해서는 레이서(layer)를 잘 활용할 필요가 있다. 'qplot()'으로는 문법을 효과적으로 활용하는 데 한계가 따른다는 점과 이 때문에 그래프를 핸들링하는 데

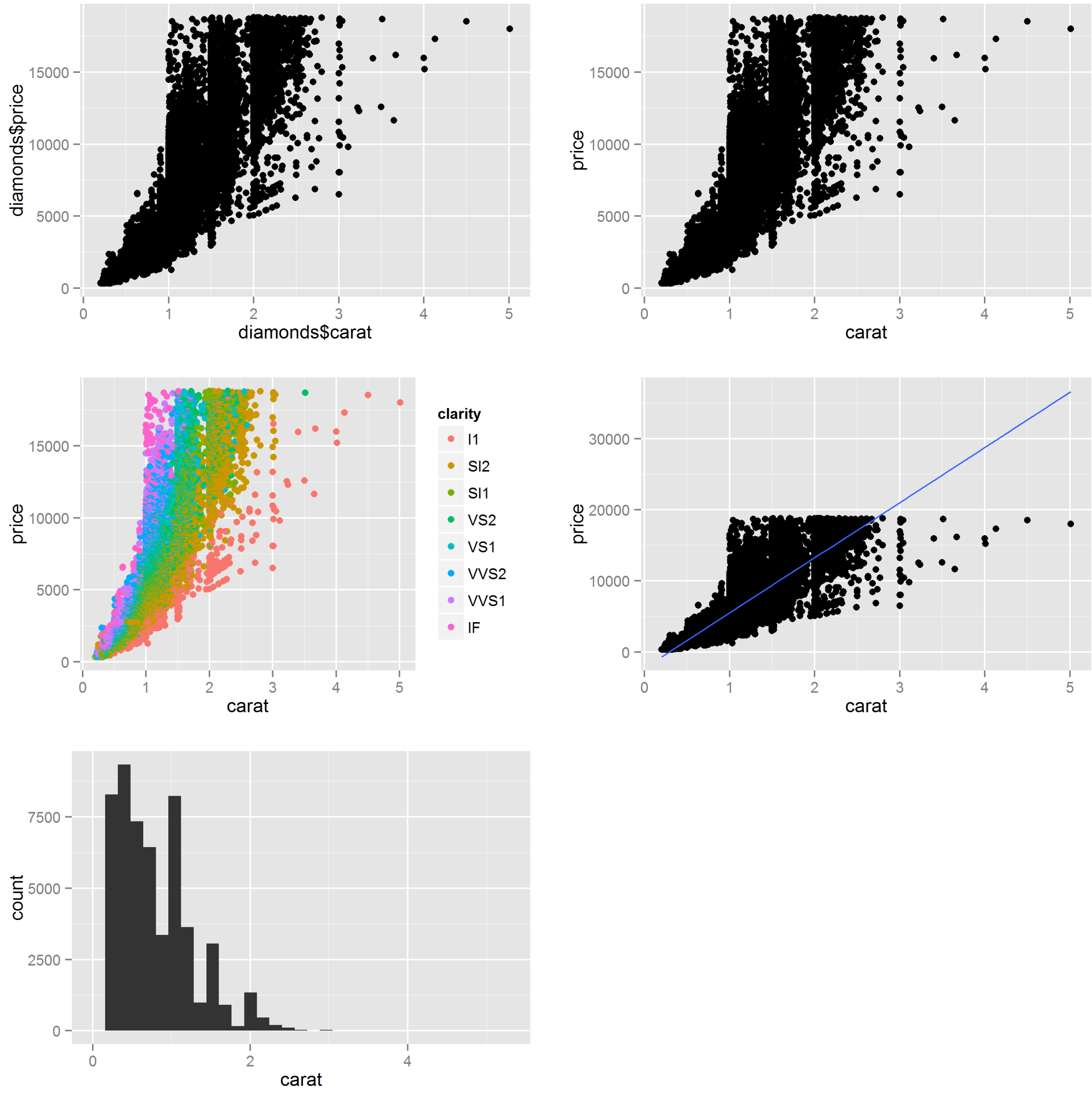

그림. 4.1: qplot 예제 결과

```
> ggplot(data=diamonds, aes(x=carat,y=price)) + geom_point(aes(colour=clarity))
```

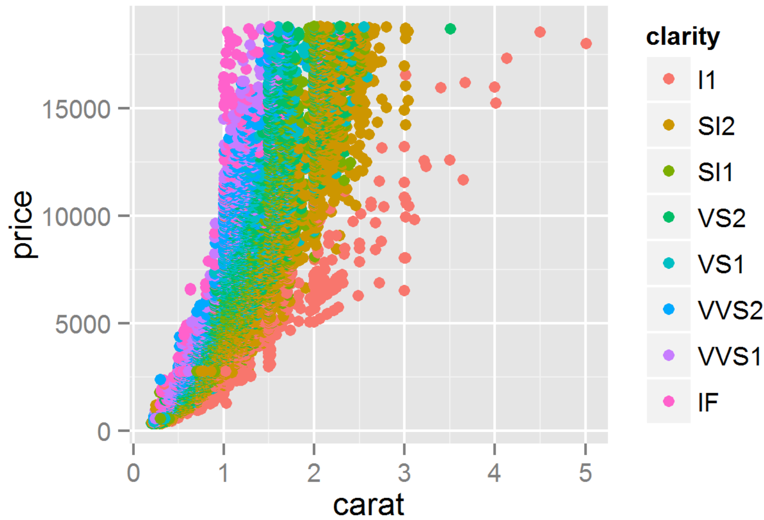

그림. 4.2: ggplot2 예제

인터페이스상의 복잡함이 존재한다는 점이다. 다른 한 가지는 'qplot()'으로는 문법을 배우고 가르치기가 어렵다는 것이다. 따라서 다음과 같은 플로팅 방식의 문법을 사용할 것이며, 물론 이 결과인 그림4.2는 그림4.1의 결과중에 하나와 같다는 것을 알 수 있다.

물론 위 코드는 동일한 그래프를 플로팅한다.

4.2 문법 (GRAMMAR OF GRAPHICS)

일단 간단한 데이터를 갖고 문법의 각 단계가 어떻게 적용되는지 살펴보자.
이 예제는 Hadley의 발표자료에서 가져왔다. 이유는 내부 데이터가 어떻게 바뀌면서 각 문법이 적용되는지 ggplot2의 내부를 들여다 보기가 여의치 않다는 점을 이해해주기 바란다.

length	width	depth	trt
2.00	3.00	4.00	a
1.00	2.00	1.00	a
4.00	5.00	15.00	b
9.00	10.00	80.00	b

Table 4.2: 문법설명 예제 데이터

Table 4.2의 length, width를 기반으로 아래와 같이 scatter plot을 그린다고 해보자.

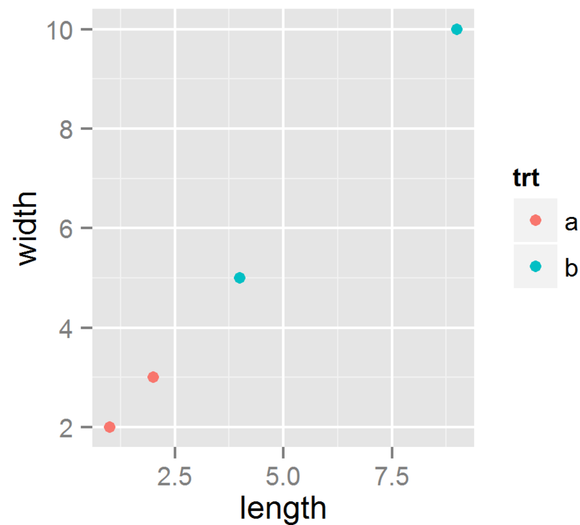

그림. 4.3: 첫번째 테스트 데이터 시각화

일단 scatter plot은 각 축에 숫자형 데이터를 입력받아 좌표계에 점을 찍는 것이다. 점은 'ggplot2'에서 기하객체(geometric object)라고 불리며, 문법에서는 짧게 geom이라고 한다. 여기에는 점(points), 선(lines), 다각형(polygons) 등 많은 객체가 존재한다.
데이터를 플로팅 하기 위해 데이터의 각 레코드를 갖고 그래픽 요소에 매핑할 필요가 있다. 이를 '미적 요소 매핑(aesthetic mapping)'이라고 표현하며, 'ggplot2'의 문법에서는 aes라는 함수가 이 역할을 한다. 'aes(x=length, y=width)' 코드가 의미하는 바는 x축에 length 컬럼을 매칭시키고, y축에 width 컬럼을 매칭시키라는 것이다.
미적 요소 매핑은 데이터의 각 속성을 그래프 속성에 매핑을 시킨다. scatter plot의 경우 좌표계 속성에 필요한 x, y, 그리고 점의 모양, 점의 크기, 점의 색깔 등이 그에 해당한다. 이를 위해 ggplot2 내부에서는 다음과 같이 Table 4.3와 같은 형태로 데이터를 변환시킨다.

x	y	colour
2.00	3.00	a
1.00	2.00	a
4.00	5.00	b
9.00	10.00	b

Table 4.3: 미적 요소 매핑 데이터

'length->x, width->y, trt->colour'로 변환된 것을 확인할 수 있다. 점의 크기나 모양은 매핑 속성에 없으므로 매핑되지 않았으나, 모두 같은 기본 값으로 암묵적으로 생성된다. 모양의

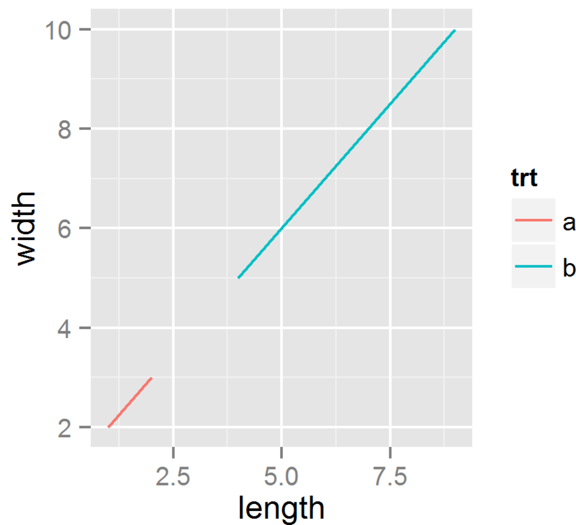

그림. 4.4: 라인차트로 테스트 데이터 시각화하기

기본값은 속이 꽉찬 점, 점의 크기는 1이 기본 값이다. 미적 요소 매핑은 어느 데이터가 어떤 곳에 쓰일지를 명시하는 작업이라고 생각하면 된다. 앞의 매핑된 데이터는 그림4.4과 같이 line chart에서도 사용될 수 있는 데이터다. 잘 생각해보면 이들 그래프를 그리기 위해 필요한 것은 x, y축에 따른 값과 색상을 구분하는 변수뿐이다. 그러나 이런 그래프는 이 상황에서는 그다지 의미가 없으니 그냥 참고하기 바란다.

미적 매핑을 한 후에 진행되는 작업은 매핑된 데이터를 갖고 컴퓨터가 알아볼 수 있는 데이터로 변환시키는 것이다. 이는 흡사 디지털 카메라 자동 프로세스를 통해 이해하면 쉬울 것이다. 디지털 카메라는 피사체를 CCD에 매핑한 후 디지털 이미지화해 메모리에 저장하는 과정으로 작동한다. 컴퓨터가 이해할 수 있는 이미지 포맷으로 변환하는 작업이 바로 'ggplot2'의 스케일링(scaling) 작업이다. x, y축 데이터는 이미지를 출력하는 대상에 맞게 변수 변환이 이뤄진다. 'ggplot2'에서 사용하는 시스템은 grid이기 때문에 `[0,1]` 사이의 값으로 스케일링 된다. 그리고 colour 값은 자동으로 사람의 눈으로 구분하기 쉬운 색상으로 매핑된다. 사람이 구분하기 쉬운 색상을 사용하기 위한 작업도 이뤄지는데 생상환(color wheel)[2]을 구분하고 싶은 레벨 개수기반으로 색상과 명암 기준으로 일정하게 분할해 색상을 매칭시킨다. 'ggplot2'는 자동으로 이 작업을 해주기 때문에 다변량에 대한 그래프를 그릴 때 매우 편하다. 물론 사용자가 이 값들을 직접 정해줄 수 있다.

스케일링을 통해 나온 데이터는 Table 4.4와 같다.

재미있게도 x, y의 스케일링된 데이터 값은 동일한다. 이것이 앞 그림 4.4에서 어떻게 표현되

[2] `http://en.wikipedia.org/wiki/Color_wheel`

x	y	colour
0.12	0.12	#FF6C91
0.00	0.00	#FF6C91
0.38	0.38	#00C1A9
1.00	1.00	#00C1A9

Table 4.4: 스케일링이 적용된 데이터

는지 비교확인해 보면 이해가 빠를 것이다.
사실 x, y의 변환된 데이터를 일반적인 직교 좌표계를 사용하거나 극 좌표계를 사용하느냐에 따라서 한번의 변환이 더 일어난다. 기본값은 직교 좌표계이기 때문에 예상대로 출력된 것을 확인할 수 있다.
이들 데이터 말고도 하나의 그래프를 그리기 위해서는 더 많은 것이 필요하다. 예를 들면, 축(axis), 레전드(legend), 레이블(lable), 그리드(grid) 라인들 등인다. 이들은 사용자가 직접 설정하지 않으면 기본값으로 출력된다. 이들을 직접 핸들링하기 위해서는 ggplot2의 테마(theme)와 관련된 매뉴얼을 찾아봐야 한다.
이상으로 개략적인 ggplot2의 동작 방식을 살펴봤다. 이 이외에 ggplot2에서 중요한 레이어(layer) 시스템이 있다. 이 레이어 시스템 덕분에 그래프 위에 다양한 정보를 추가해 그래프를 더욱 '정보성 있게' 만들 수 있다.

```
> ggplot(test.data, aes(x=length, y=width)) +
>   geom_point(aes(colour=trt)) + geom_smooth()
```

위 코드에서 '+' 이후의 geom_*부분은 좌표계에 레이어를 덧씌우는 역할을 한다. 'ggplot2'에서는 직관적으로 '+' 연산자를 사용해 이를 연동한다.
이런 레이어를 플롯에 올릴 때 원 데이터는 다음과 같은 변환 과정을 통해 레이어 개별적으로 적용받는다.

1. 미적 매핑
2. 통계적인 변환(stat)
3. 기하객체에 적용(geom)
4. 위치 조정(position adjustment)

1번과 2번 사이에 스케일링이 작업이 들어간다는 것을 명심하자. 4번 위치 조정의 경우 동일 좌표계에 여러 종류의 그래프를 올려놓기 위해 개별 레이어의 위치 정보에 대해 학습(train)을

하는 과정이 포함된다.
만일 이 과정이 없다면 특정 레이어에 특화된 스케일만 적용돼 적절하지 못한 그래프가 출력될 것이다.

4.2.1 레이어를 이용한 ggplot2 시각화

이제 기본적인 내용은 다 설명했으니 실제 예제를 통해 알아보자. 이제부터 본격적으로 앞서 설명한 diamonds 데이터를 사용하겠다. 다음 코드는 앞서 소개한 코드다.
아래 두 코드는 같은 그래프를 그린다. 사실 qplot() 은 '+' 연산자 없이도 그래프 한장을 그리고 아래 코드의 (2) 는 앞 부분만 실행하면 그래프가 생성되지 않는다. 이유는 그래프를 플로팅하기 위해서는 최소 하나 이상의 레이어가 있어야 하기 때문이다. 아래 코드의 (2) 의 'ggplot()' 부분은 단지 데이터와 미적 요소 매핑을 하는 메타 데이터만을 생성한다. 실제 geom_point() 가 비로소 레이어 하나를 추가 하는 코드다.

```
> qplot(carat, price, data = diamonds, geom="point" ,colour=clarity) #--(1)
> ggplot(data=diamonds, aes(x=carat,y=price)) + geom_point(aes(colour=clarity))#--(2)
```

'ggplot2' 는 객체를 저장하고 불러오는 작업이 원활하게 작동한다. 따라서 다음과 같은 코드를 출력해 보는 것은 각 코드 조각이 어떻게 작동하고 어떤 데이터를 갖고 있는지 확인하기 위한 좋은 습관이다.

```
> s <- ggplot(data=diamonds, aes(x=carat,y=price))
> summary(s)

## data: carat, cut, color, clarity, depth, table, price, x, y, z [53940x10]
## mapping:  x = carat, y = price
## faceting: facet_null()
```

위 코드는 (2) 번 코드의 앞부분의 실행 결과로 나온 객체에 summary() 함수를 실행한 결과이다. 'x=carat, y=price' 와 같은 미적 요소 매핑을 한 정보가 있는 것을 알 수 있다.
필자가 'ggplot2' 에서 가장 좋아하는 특징을 설명할 때가 온 것 같다.
(2) 번 코드에서 미적 요소 매핑을 시킨 특징은 '+' 연산자로 레이어를 추가하더라도 해당 레이어에 상속된다는 것이다. 따라서 반복적인 코드 작성을 최소화할 수 있다. 물론 상속 내용을 쓰지 않고 재정의 (overriding) 해 사용할 수 있다. 아예 매핑 정보를 무효화할 수도 있다. 하지만 대부분 같은 데이터를 활용해 한 플롯에 중복적으로 시각화하는 경우가 많아 이 부분이 상당히 유용하다. 따라서 (2) 번의 geom_point 부분의 코드는 실제 다음 코드와 같다.

```
> ... + geom_point(aes(x=carat, y=price, colour=clarity))
```

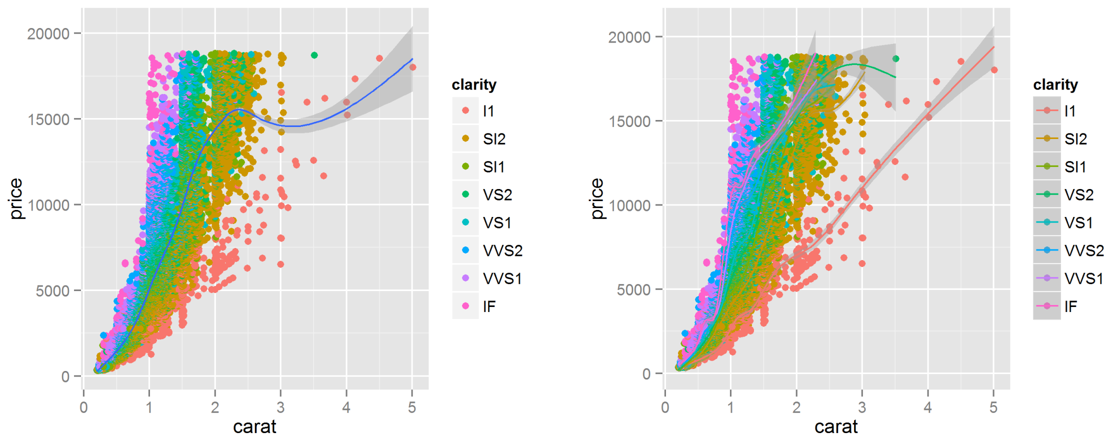

그림. 4.5: 요소를 상속한 레이어(좌)와 재정의한 레이어(우) 그래프, 같은 결과를 보여준다.

추가된 레이어에서 미적 요소 매핑을 한 코드는 그 다음 레이어로 상속되지 않는다. 그 차이를 알 수 있는 코드는 다음과 같다.

```
> ggplot(data=diamonds, aes(x=carat,y=price)) +
>   geom_point(aes(colour=clarity)) + geom_smooth()  # ---(1)
>
> ggplot(data=diamonds, aes(x=carat,y=price, colour=clarity)) +
>   geom_point() + geom_smooth()                      # ---(2)
```

위 결과 그래프는 그림4.5과 같다.

앞의 코드 (1)은 'ggplot()'에서 행한 매핑이 하부 레이어에 상속이 됐음을 보여준다. 더불어 'geom_point()'에서 정의된 colour매핑 정보는 'geom_smooth()'에 전달되지 않았음을 확인할 수 있다. 하지만 (2)의 경우 colour까지 매핑시켰더니 의도된 대로 clarity에 따른 회귀 곡선이 그려진 것을 알 수 있다.

'ggplot2'에서는 매핑 작업만 소개했는데, 설정(set) 옵션도 제공한다. 위 코드에서 colour 인자에 clarity 명목형(nominal) 값을 매핑해 색깔을 입혔다. 단일 색상으로 다음과 같이 설정할 수도 있다.

출력 결과는 그림4.6와 같다.

그래프의 포인트가 모두 청색으로 찍힌 것을 활인할 수 있다. 물론 'ggplot()'에서 colour 매핑을 시켰지만 geom_point를 그리는 레이어에서 다시 색상을 설정해 그 색상이 출력된 것이다.

이로써 많은 미적 매핑 요소를 살펴봤다. 이 요소들 중에서 아직 소개하지 않은 group 매핑 요소를 알아보자. 사실 이 요소는 colour 매핑 요소와 매우 유사한 성격을 갖고 있다. colour

```
> ggplot(data=diamonds, aes(x=carat,y=price, colour=clarity)) + geom_point(colour='darkblue')
```

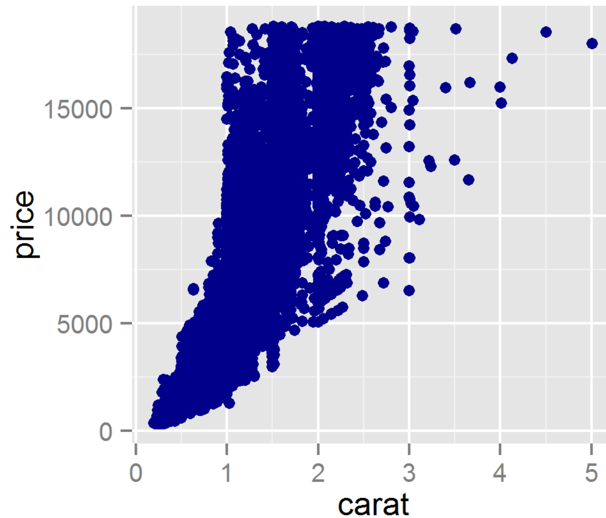

그림. 4.6: colour에 단일값 설정하기

요소 역시 데이터는 colour에 매핑된 데이터 기준으로 쪼개 각 데이터 그룹마다 서로 다른 색상을 입히는 역할을 한다. group 역시 기준값을 갖고 소그룹으로 나눠 각 요소에 stat과 geom을 적용시키는 역할을 한다.
일단 다음 코드를 보면서 알아보자.

```
> p <- ggplot(data=diamonds, aes(x=carat,y=price))
> p + geom_smooth()                          # --- (1)
> p + geom_smooth(aes(group=clarity))    # --- (2)
```

그룹 매핑을 하지 않은 앞 코드의 (1)은 전체의 데이터를 하나의 회귀 곡선에 피팅시킨 그림4.7 왼쪽의 그래프를 출력한다. group 매핑을 한 (2)는 각 clarity 그룹별로 데이터를 나눠 각기 별도의 회귀곡선을 피팅시킨 라인을 그림4.7의 오른쪽 그래프와 같이 출력한다.

4.2.2 GEOM

기하객체(geometric object)는 실제 레이어를 랜더링하는 역할을 한다. 각 기하객체는 그에 맞는 미적 매핑 요소를 필요로 한다. 예를 들어 point 기하객체의 경우 x, y 미적 요소를 필요로 하며, geom_bar 객체의 경우 ymax 값과 더불어 테두리 색상과 내부 채움 색상을 필요로 한다.

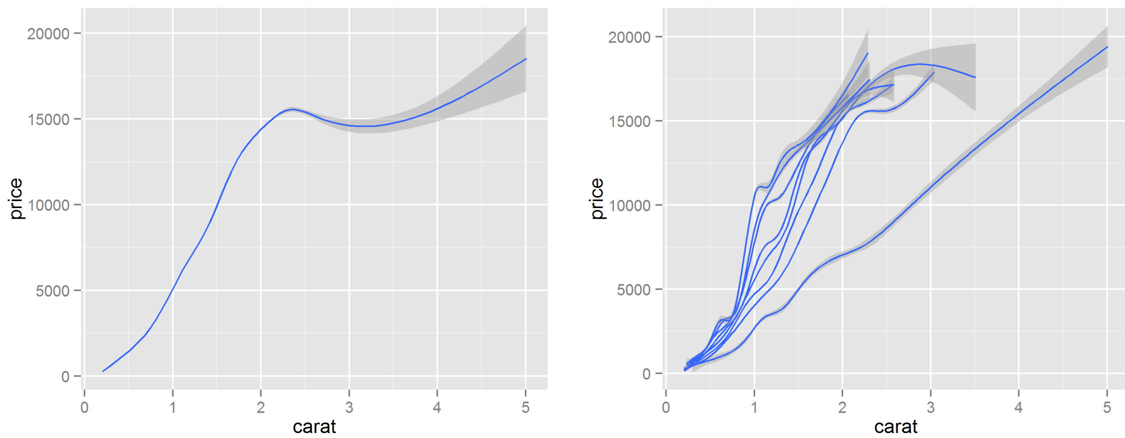

그림. 4.7: 그룹매핑 유무에 따른 그래프 결과 차이

4.2.3 STAT

앞서 geom의 경우 그에 맞는 데이터가 필요하다고 소개했다. stat은 (stat에 기본적으로 적용되는) geom에 필요한 데이터를 주어진 데이터에서 생성하는 역할을 한다. 예를 들어 stat_bin의 경우 histogram을 그리기 위한 통계계산 작업을 통해 다음과 같은 데이터를 갖는 'data.frame'을 출력한다

- count: 각 빈(bin)에 해당하는 관측값의 개수
- density: 각 빈(bin)의 밀도(전체의 합이 1이 된다.)
- ncount: count와 같으나 `[0,1]`로 스케일링 된다.
- ndensity: density와 같으나 값의 범위가 `[0,1]`로 스케일링 된다.

기본값으로는 count를 가지고 histogram을 그리게 된다.
아래의 코드와 그림4.8를 참고해서 stat_bar에서 생성된 데이터를 어떻게 쓰는지 확인해 보자!

```
> ggplot(data=diamonds, aes(x=price)) + geom_bar() # --(1)
> ggplot(data=diamonds, aes(x=price)) + geom_bar(aes(y=..count..))  # --(2)
> ggplot(data=diamonds, aes(x=price)) + geom_bar(aes(y=..density..))
> ggplot(data=diamonds, aes(x=price)) + geom_bar(aes(y=..ncount..))
> ggplot(data=diamonds, aes(x=price)) + geom_bar(aes(y=..ndensity..))
> ggplot(data=diamonds, aes(x=price)) + geom_bar(aes(y=..density..)) + ylab("밀도")
```

그림에서 보듯이 stat에서 생성된 data.frame의 필드를 사용하려면 '..' 기호를 쓰면 된다. 이는 주어진 원본 'data.frame'의 데이터 필드 이름과 혼용되는 것을 피하기 위함이다.

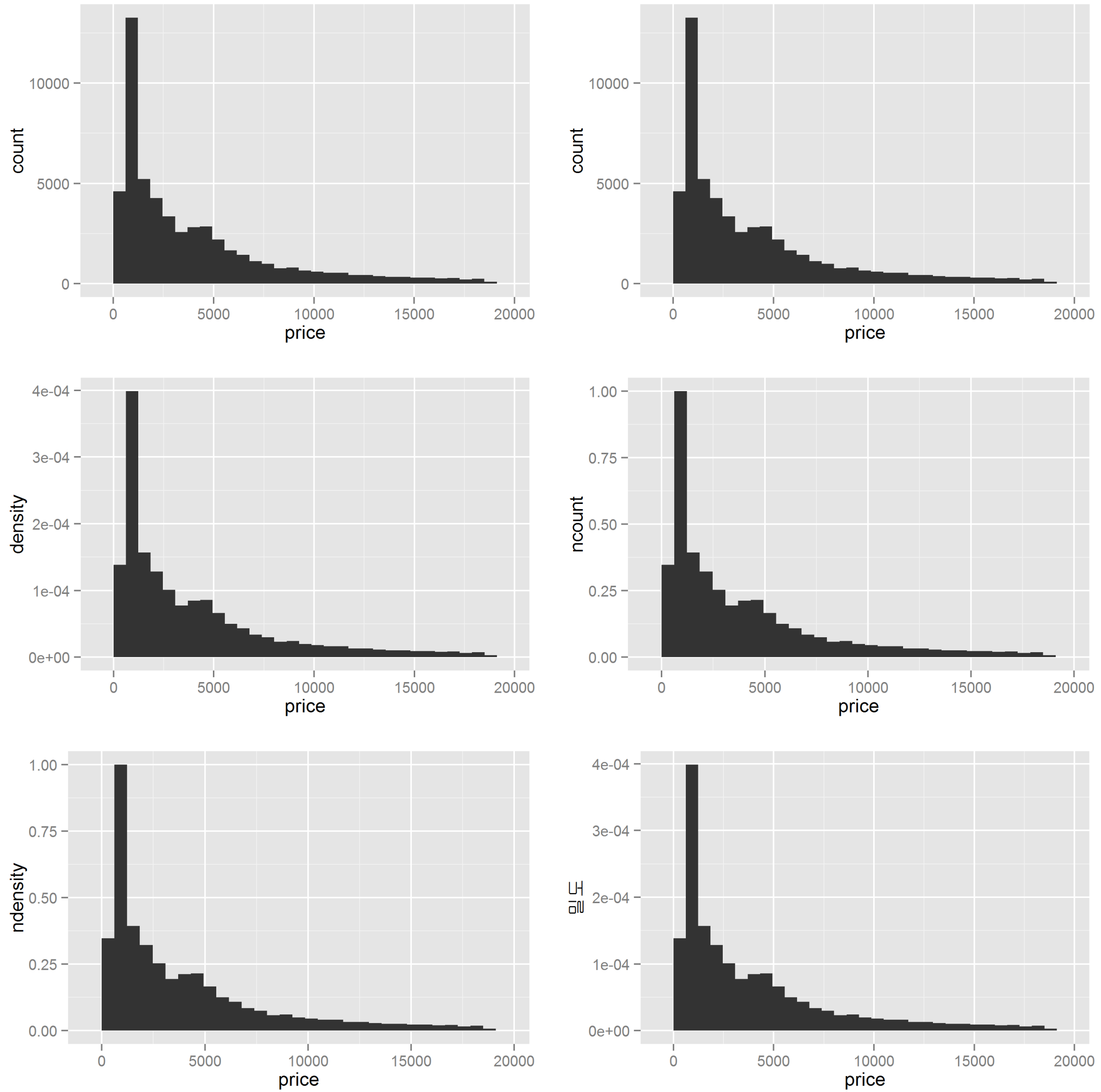

그림. 4.8: stat_bar의 결과 값에 따른 그래프. (1), (2) 코드가 차례로 제일 위의 그래프를 출력하고, 나머지는 y축의 이름으로 구분해 보면 된다.(위에서 아래로, 왼쪽에서 오른쪽으로 순서다)

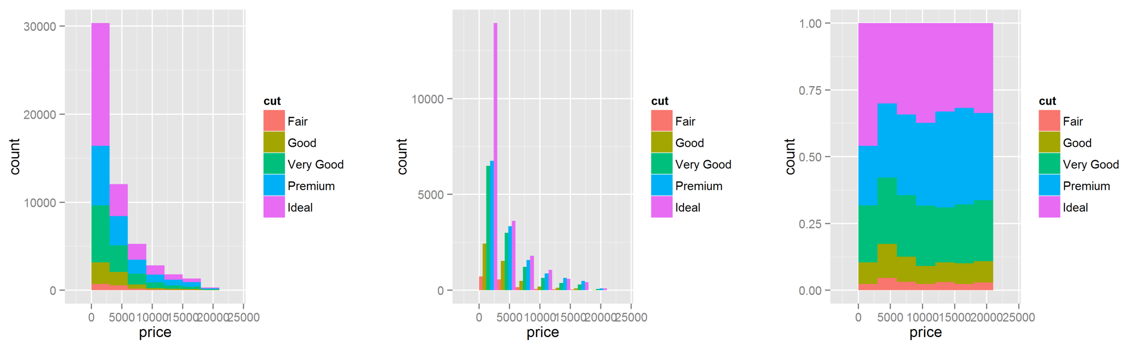

그림. 4.9: ‘identity’, ‘dodge’, ‘fill’ 옵션에 따른 그래프

geom과 stat 객체들은 먼저 소개했던 링크[3]에 정리돼 있으니 참고 바란다.

4.2.4 위치 조정

위치 조정의 경우 이산형 값에 대해 주로 행해진다. 필자의 경우 막대그림(bar plot)이나 히스토그램(histogram)을 그릴 경우 이 기능을 주로 사용한다. 백 마디 말보다 그림 한장이 더 도움이 될 수 있으니 다음 코드를 보자.

```
> ggplot(data=diamonds, aes(x=price)) +
>   geom_bar(aes(fill=cut), binwidth=3000)
> ggplot(data=diamonds, aes(x=price)) +
>   geom_bar(aes(fill=cut), binwidth=3000, position="dodge")
> ggplot(data=diamonds, aes(x=price)) +
>   geom_bar(aes(fill=cut), binwidth=3000, position="fill")
```

위 두 번째 그래프와 유사한 결과는 facet이라는 기법으로 표현이 가능하다. facet이라는 기법은 데이터를 특정 기준에 따라 서브셋으로 나눈 뒤 각 서브셋을 각기 다른 그래프 패널에 출력하는 것을 의미한다. 따라서 위의 두번째 그래프는 다음 그림4.10과 같이 표현할 수 있다.

비슷한 역할을 하는 face_wrap이라는 함수도 있다.

4.2.5 GEOM과 STAT의 결합

geom과 stat은 다양한 방법으로 결합될 수 있다. 물론 어떻게 결합이 될 수 있는지 그리고 목표로 하는 geom에 필요한 통계 데이터가 무엇인지는 확인해야 한다. 자세한 내용은 메뉴얼을 참고하고 다음 같이 다양한 그래프가 같은 stat을 기반으로 도출될 수 있음 염두에 두길 바란다.

[3] http://docs.ggplot2.org/current/

```
> ggplot(data=diamonds, aes(x=price)) + geom_bar(binwidth=3000) + facet_grid( . ~ cut)
```

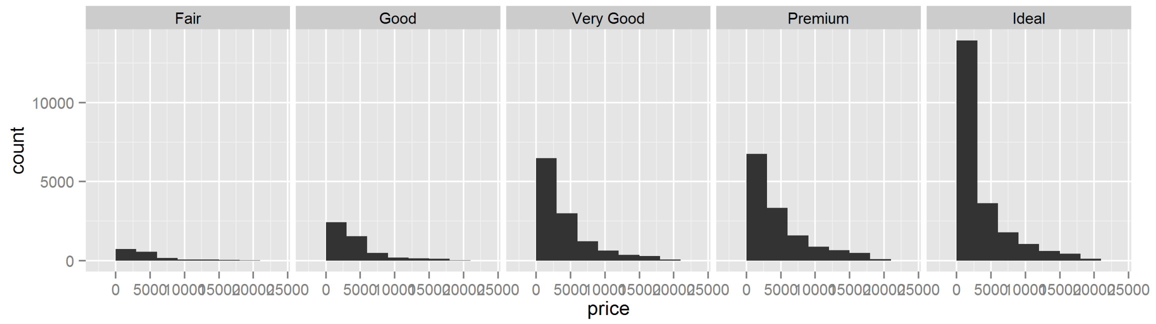

그림. 4.10: facet 결과 그래프

```
> ggplot(data=diamonds, aes(x=price)) +
+   geom_bar(binwidth=3000) + facet_wrap( ~ cut, nrow=3)
```

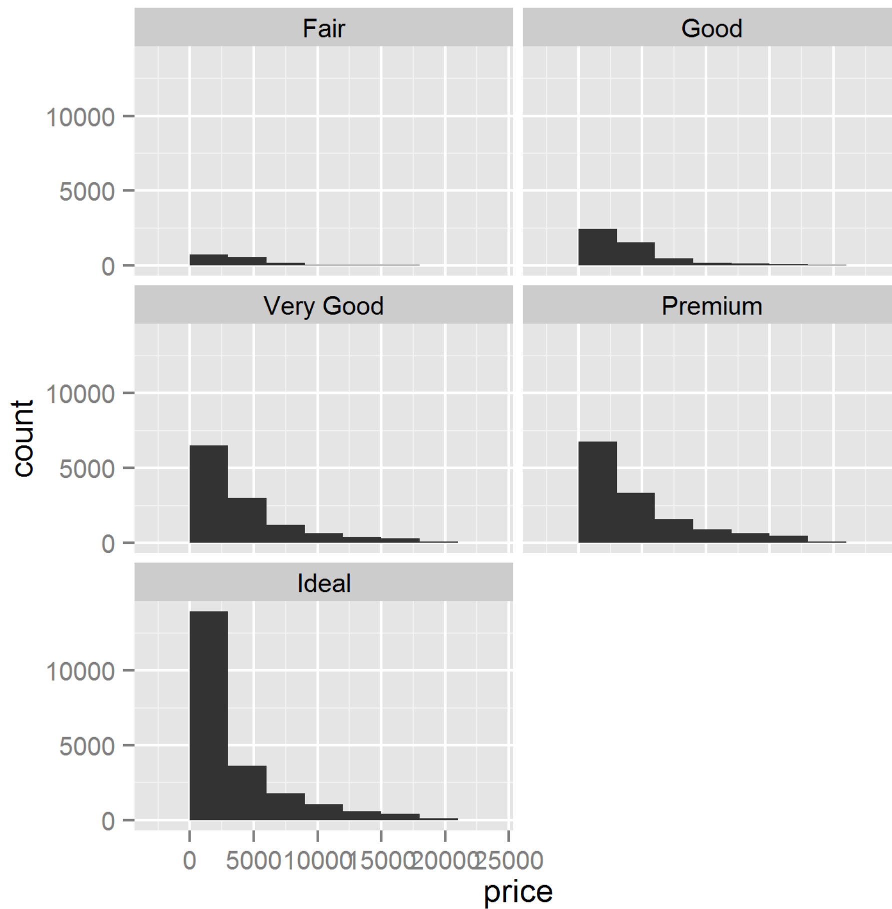

그림. 4.11: facet_wrap 결과 그래프

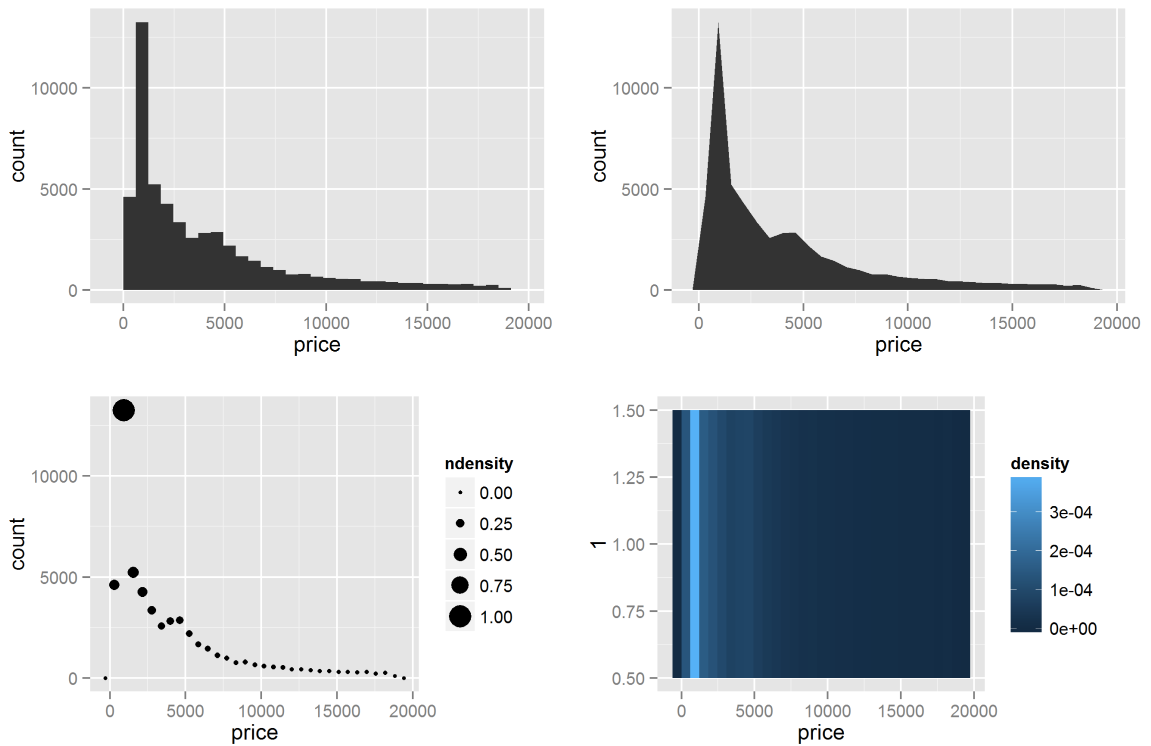

그림. 4.12: geom과 stat을 결합한 다양한 그래프 표현 방식

```
> d <- ggplot(diamonds, aes(price))
> d + stat_bin(geom="bar")
> d + stat_bin(geom="area")
> d + stat_bin(aes(size=..ndensity..),geom="point")
> d + stat_bin(aes(y=1, fill=..density..),geom="tile")
```

곰곰이 위와 같은 그래프가 어떻게 그려질까 하고 생각해보자. 이것을 그리기 위해서 어떤 데이터와 가공이 필요한가를 생각해보면 더욱 다양한 그래프를 생성할 수 있다.

4.3 마지막 예제

이제 마지막 예제를 제시해야 될 시점이다. 실제 지금까지 설명한 내용만으로도 수많은 그래프를 만들어 볼 수 있다. 하지만 실무에서 특정 데이터를 갖고 원하는 그래프를 만드는 것이 그렇게 말처럼 쉽지만은 않다. 따라서 그런 경우를 가정해 그래프를 그려보는 예제를 제시한다.

우리가 마지막으로 쓸 예제 데이터는 통계청에서 제공하는 산업생산지수다. 이 산업생산지수는 다음과 같은 의미를 갖고 있는 국가 통계 데이터다.

> 2007년까지는 산업생산지수라 표현하였고, 현재 광공업생산지수라고 표현하고 있다. 광공업생산지수는 다종 다양한 생산활동의 결과인 작업량(work done)을 측정하여 전체 광업, 제조업, 전기,가스업 생산의 수준과 변동추이를 알기 위해 작성한다. 이 지수는 광업, 제조업, 전기,가스업의 생산활동 동향, 경기동향을 알 수 있게 하는 기본적인 경기지표이다. 생산지수에서 '생산'이라는 개념은 국내총생산(GDP)에서의 생산(부가가치 개념)이라는 개념과 같다. 광공업지수가 포괄하고 있는 산업의 경제활동은 전체 경제활동의 약 30%를 차지하고 있다. 따라서 광공업생산지수는 전체 경제의 생산활동의 움직임을 월별로 판단할 수 있는 지표이다. 더구나 월별 변화를 신속히 나타내는 속보성 외에도 경기에 민감하며, 실질경제의 움직임을 나타내는 지표 중 가장 공표가 빠른 지표이다.

자료 출처는 통계청이며, 실습 목적상 데이터는 저자가 온라인으로 제공하겠다.
우리의 목적은 산업생산지수의 원계열과 계절조정계열의 데이터를 x를 시간축으로 하는 시계열 도표로 표현하는 것이다. 이곳에 여러 정보를 가미한 텍스트와 색상을 입히는 작업을 시도할 것이다.
일단 기본 플로팅을 해보도록 하자!

보다시피 x, y축의 레이블이 적당하지 않으며, 오른쪽 레전드(legend)의 제목/순서도 적절하지 않고 시간 간격도 너무 먼 것을 확인 수 있다. 일단 x, y축 레이블과 레전드 제목을 바꿔보자.

scale_color_hue 함수는 사실 기본 옵션으로 사용되는 함수다. colour 미적 요소 매핑을 한 정보에서 제목을 재정의하기 위해 호출된 함수다. 보듯이 첫 번째 인자가 스케일링되는 레전드의 제목이 된다.
scale_* 유의 함수가 ggplot2에서는 상당히 많다. 이는 미적 요소 매핑을 한 결과를 어떻게 화면에 출력할지 방법을 적용하는 함수이기에 alpha, fill, shape, size와 같은 미적 요소를 사용할 때에는 scale을 어떻게 할지 위와 같은 함수를 찾아볼 필요가 있다.

scales 패키지는 'data_format' 함수를 사용하기 위해 불러들였다. 위의 코드는 2년 간격으로

```
> load(url("http://dl.dropbox.com/u/8686172/ipidf.RData"))
>
> ipidfg <- ggplot(ipidf, aes(ipidf.date)) +
+   geom_line(aes(y=ipidf.ipi.sa, colour="산업생산지수(제조업)-계절조정-")) +
+   geom_line(aes(y=ipidf.ipi, colour="산업생산지수(제조업)-원계열-") )
> ipidfg
```

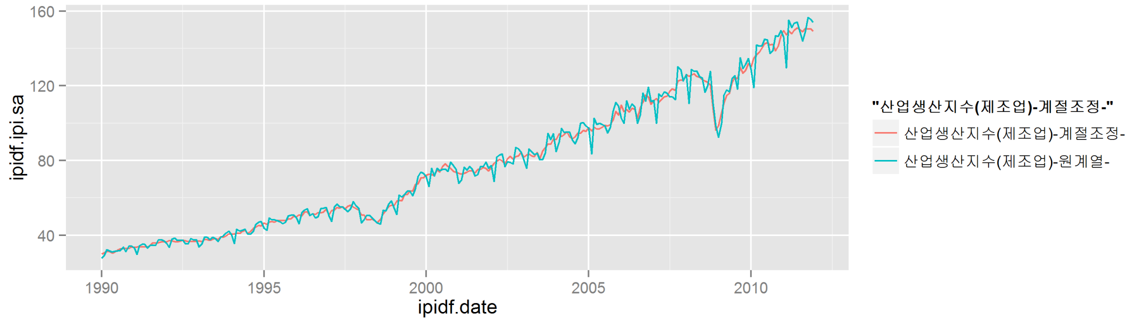

그림. 4.13: 산업생산지수 그래프

```
> ipidfg2 <- ipidfg + xlab("날짜") + ylab("생산지수") +  scale_color_hue("산업생산지수")
> ipidfg2
```

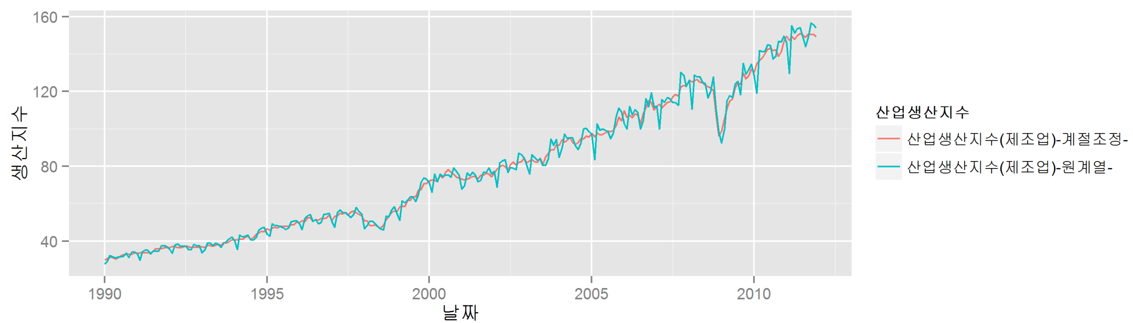

그림. 4.14: 레전드 제목/순서가 개선된 산업생산지수 그래프

```
> library(scales)
> ipidfg3 <- ipidfg2 + scale_x_date(breaks="2 years", labels = date_format("%Y-%m")) +
+   guides(colour = guide_legend(reverse=TRUE))
> ipidfg3
```

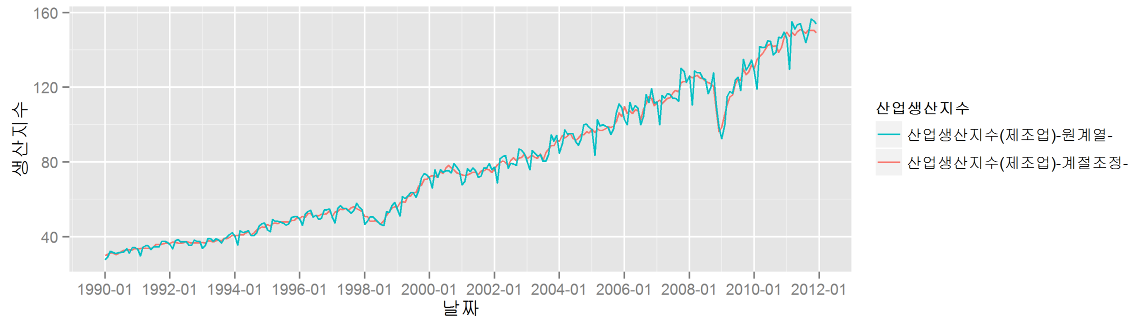

그림. 4.15: 날짜 형식과 레전드 색상이 뒤바뀐 산업생산지수 그래프

x축 레이블의 날짜를 출력하는 역할을 하며, guide 함수는 각 스케일이 적용되는 요소에 옵션을 재정의해 주는 역할을 한다. 물론 레전드의 제목을 설정하기 위해 scale_color_hue를 사용한 것을 빼고 guide를 다음과 같이 정의해도 된다. 'reverse=TRUE' 옵션은 레전드의 출력 순서를 바꾸기 위함이다. 이 부분은 그림4.15를 참고하기 바란다.

```
> ... + guides(colour = guide_legend('산업생산지수', reverse=TRUE))
```

그런데 그래프 x축을 보면 텍스트를 표현하기에 너무 좁은 것을 알 수 있다. 물론 그래프의 x축 너비를 늘리면 되겠지만, 그것은 이 w장의 교육 목적상 적절하지 않으므로 x축의 텍스트를 90도 시계 반대 방향으로 돌려서 표현하겠다. 이를 위해 'theme' 함수를 사용하는데, 이 함수는 ggplot2의 테마(theme) 시스템의 상세 설정을 하는 함수다. 따라서 x축의 텍스트 회전도 역시 'ggplot2'에서는 테마의 영역에 해당하는 부분이라 할 수 있다. 아쉽게도 테마에 대한 내용은 이 강좌의 교육 범위를 넘어가므로 자세한 내용은 앞서 소개한 책을 참고하기 바란다. 여기서는 간략히 코드 소개만 한다.

4.4 장을 마치며

지금까지 'ggplot2'에 대한 많은 내용을 살펴봤다. 하나부터 열까지 모두 한꺼번에 다 자동으로 되는 패키지였으면 좋겠지만, 생각보다 원하는 바를 도출하려면 손봐야 할 것들이 많음을 실감했을 것이다. 대부분 유연함과 난이도는 서로 정비례한다. 그러나 유연함과 난이도의 많은 부분을 문법이라는 것으로 정리해 놓았다는 게 이 패키지의 가장 큰 장점이다.
그동안 연재 중에서 가장 길게 설명했다. 이마저도 이 패키지를 설명하는 데 부족하다. 세세한

```
> ipidfg3 + theme(axis.text.x=element_text(angle=90, hjust=1))
```

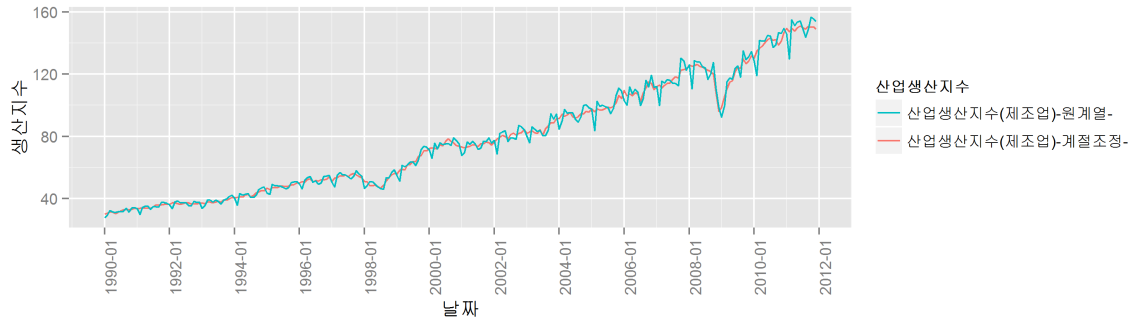

그림. 4.16: theme를 이용한 x축 텍스트 조정

부분을 설명하기 위해서는 책 한 권 정도의 분량이 필요할 것인데, 그마저도 모든 옵션을 설명하기엔 부족하다. 따라서 이번 연재를 통해 그래프를 그리는 감을 얻어 매뉴얼과 인터넷의 정보를 기반으로 하나하나 섭렵해 나가길 바란다.

Chapter 5

잉크스케이프를 활용한 그래프 후처리

프로그래머에 가까운 필자가 그래픽 후처리 프로그램을 사용한다는 게 이상하기도 하다. 물론 필자 역시 1년 전만 해도 이런 원고를 쓸 줄은 예상도 하지 못했다. 하지만 이번 글은 R 기반의 데이터 시각화라는 전체 주제를 마무리 짓기 위한 것으로서, 독자들에게 이런 마지막 과정도 있다는 것을 알리는 것 만으로도 의미가 있다고 생각한다.
지난 장에서 제공한 ggplot2에 대한 옵션을 더 발전시키면 이미지 출력에 대한 더 세밀한 컨트롤이 가능하긴 하지만, 많은 데이터 시각화 업무를 하는 분들은 대략의 작업을 엑셀, R에서 한 다음에 매끈하게 다듬는 일을 어도비 일러스트레이터에서 한다. 사실 일러스트레이터를 이용해 처음부터 끝까지 그래프 작업을 할 수 있다. 하지만 굉장히 많은 수작업이 들어가기 때문에 단순 작업의 연속이 되며, 그에 따라 실수도 많아진다. 그래서 필자는 주로 R의 ggplot2로 기본적인 그래프 출력을 한 뒤 폰트 및 텍스트 작업을 추가적으로 수행한다. 물론 필자는 비싼 어도비일러스트레이터를 사용하기 보다는 무료인 잉크스케이프(Inkscape)[1]를 사용하지만 말이다.
이번 글에서는 딱 필자가 사용하는 수준의 그래프 작업만을 소개하겠다. 잉크스케이프도 역시 처음 사용하는 사람에게는 만만한 그래픽 툴이 아닌지라 이 툴에 대한 자세한 내용은 관련 소개서를 참고하기 바란다.

5.1 환경 설정하기

리눅스, OS X, 윈도우. 이 대표적인 세 가지 운영체제에서 R이 주로 사용되나, 이번 글은 윈도우를 주요 플랫폼으로 사용하겠다. 이는 R에서 벡터 그래픽을 출력하기 위한 명령어인 pdf의 폰트 설정이 서로 다르기 때문이다. 폰트는 '맑은고딕'을 사용한다. 이 폰트는 윈도우 운영체제에 기본 탑재됐고, 이 폰트가 없는 윈도우7 이전의 OS 사용자는 직접 설치하기 바란다.
사실 윈도우에서든 맥에서든 리눅스에서든 한글이 포함된 pdf를 출력한다는 건 쉬운 문제가

[1] `http://inkscape.org/`

아니다. 그나마 R에서는 2.15 이후 cairo[2]가 기본 탑재돼 정말 쉬워졌다.

일단 잉크스케이프 윈도우 버전을 `http://inkscape.org/download/?lang=en`에서 내려 받아 설치한다. 윈도우 Installer 버전으로 설치하면 간편하며, 설치에 그다지 어려운 점은 없을 것이다.

5.2 예제 그래프 만들기

일단 예제로 쓸 그래프는 3장 'Data Munging With R'의 마지막에 제안했던 그래프를 사용하도록 하겠다. 이 그래프는 사실 독자들에게 출력물만 제공했으며, 코드는 제공하지 않았다. 이는 독자들에게 생각해볼 시간을 부여하기 위해서였다. 이제 다음과 같이 코드를 공개한다.

```
> library(plyr)
> library(ggplot2)
>
> load(url("http://dl.dropbox.com/u/8686172/market_price.RData"))
>
> res <- ddply(market_price, .(M_TYPE_NAME, A_NAME), summarise, avg=mean(A_PRICE))
>
> res.plot <- ggplot(res, aes(A_NAME, M_TYPE_NAME)) +
>             geom_tile(aes(fill=avg)) +
>             theme(axis.text.x= element_text(angle=90, hjust=1)) +
>             scale_fill_gradient(low="green", high="red")
> res.plot
```

기본 R에서 앞 코드를 실행하면, 그림5.1 그래프가 새창으로 뜰 것이다. RStudio에서 실행하면 오른쪽 빈 공간에 그래프가 출력될 것이다. 기본 R에서는 [파일 - 다른 이름으로 저장]을 통해 다양한 포맷으로 그래프를 저장할 수 있다.

우리가 필요로 한 포맷은 일러스트레이터나 잉크스케이프에서 쓰일 벡터 기반의 그래프로, R에서는 pdf 출력을 통해 이 파일을 얻을 수 있다(물론 SVG 포맷도 가능하지만 이 역시 한글 문제가 있다). 아쉽게도 우리가 필요로 한 pdf로 출력을 하면 그림5.2와 같이 출력될 것이다. 그 이유는 한글을 표현할 인코딩 문제 때문으로, 이것을 피하기 위해 cairo를 사용하는 것이다.

또 한 가지 문제는 폰트 처리로서 'cairo_pdf'의 기본 폰트는 한글을 표현할 수 없다. 따라서 폰트 지정을 하지 않고 출력하면 그림5.3과 같이 폰트가 깨진 그래프가 출력된다.

따라서 아래와 같은 명령어를 사용해 R이 이해할 수 있도록 폰트를 등록하자. 기본 'windowsFonts()' 명령어를 사용하면 현재 R 시스템에 등록된 폰트 매핑 테이블이 출력된다.

```
> #R 시스템에 등록된 폰트 리스트를 리스팅한다.
> windowsFonts()
```

[2] `http://www.cairographics.org/`

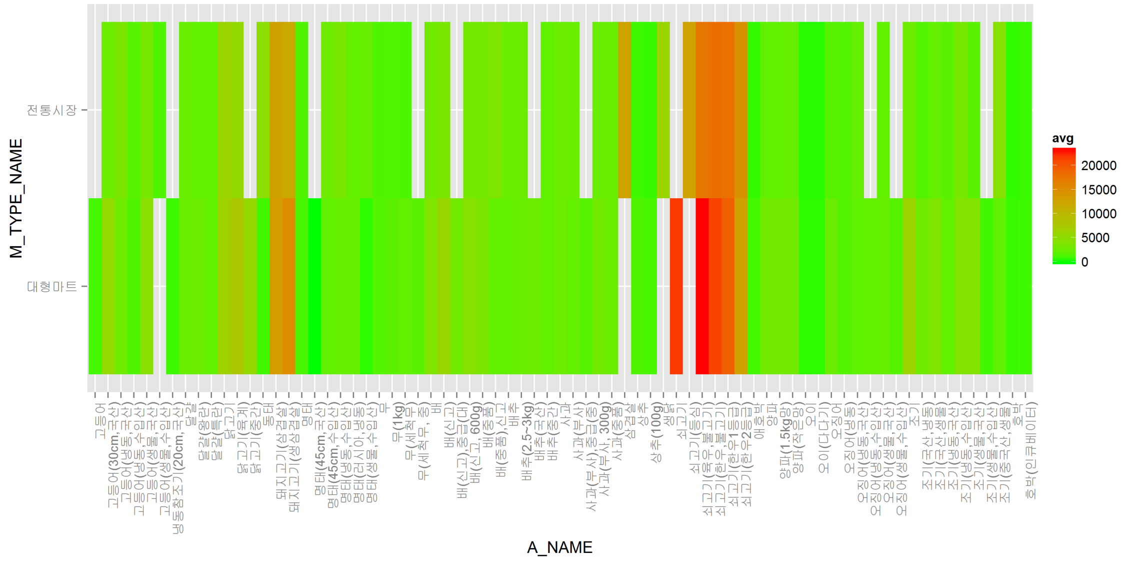

그림. 5.1: 예제로 쓰일 이미지 생성

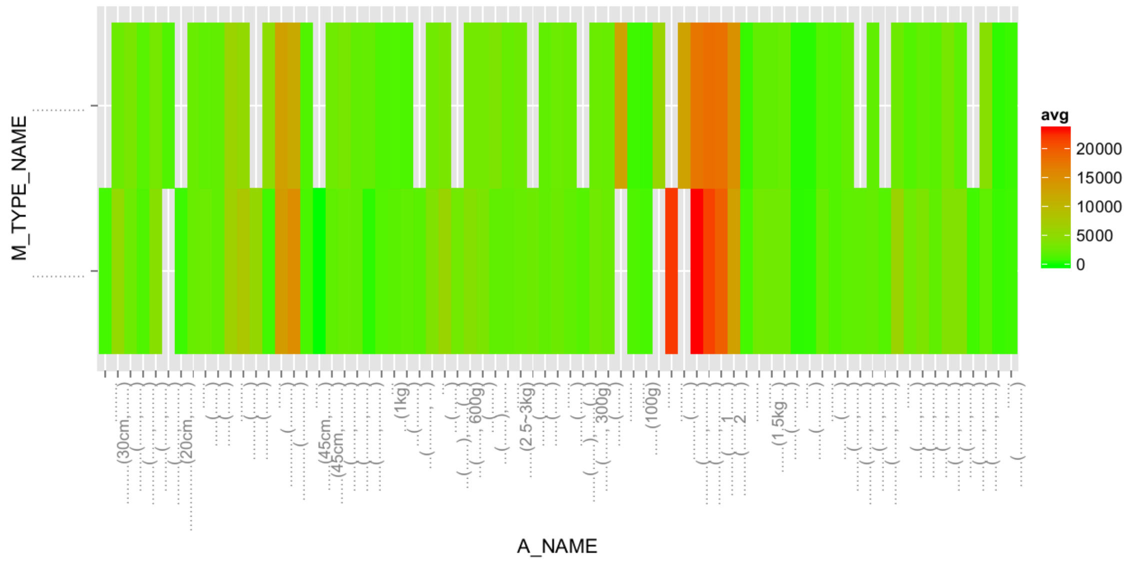

그림. 5.2: 깨진 한글 폰트

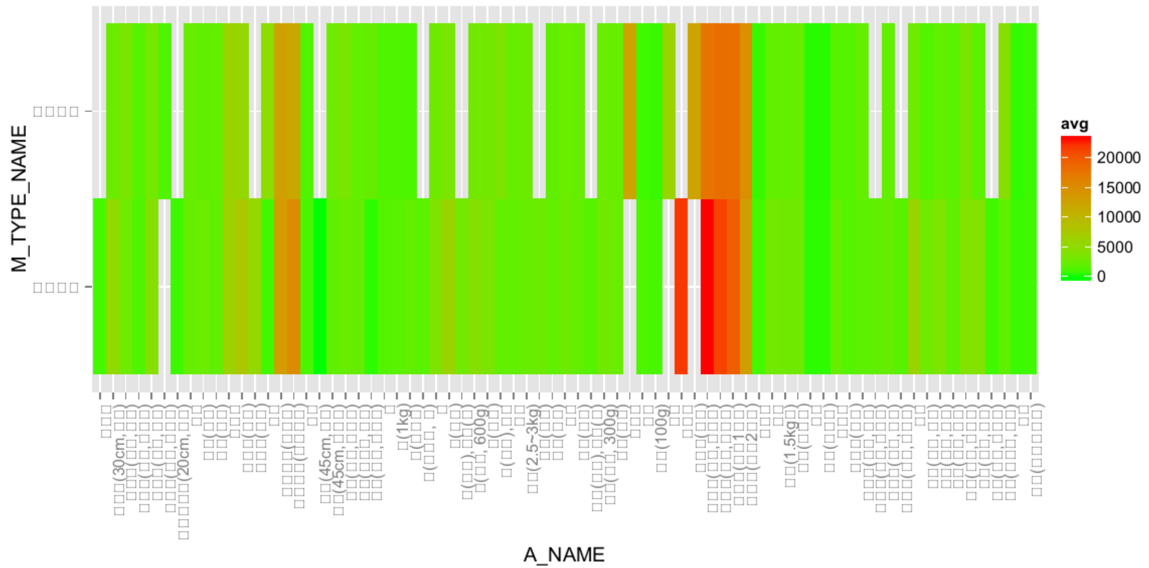

그림. 5.3: 깨진 한글 폰트(2)

```
> #폰트 등록
> windowsFonts(malgun=windowsFont("맑은 고딕"))
> windowsFonts()
> #맑은고딕 폰트를 지정하고 그래프 pdf 저장
> cairo_pdf(family="malgun",width=10,height=5)
> res.plot
> dev.off()
```

이렇게 하면 ‘getwd()’ 명령어로 출력되는 디렉터리에 ‘Rplotxxx.pdf’ 라는 이름으로 파일 하나가 생성된 것을 볼 수 있다. 우리가 의도한 그래프가 pdf로 만들어진 것이다. 이제 우리가 필요로 하는 pdf 파일은 완성됐다.

5.3 잉크스케이프로 그래프 후처리하기

일단 잉크스케이프를 실행한 후 [File - Open] 메뉴를 클릭해 만들어 놓은 pdf 파일을 읽어 들인다. 처음 읽어 들이는 데 시간이 걸리므로 참을성 있게 기다려보자. 드디어 그림5.4와 같은 잉크스케이프 화면을 볼 수 있다.

잉크스케이프에서 [F1] 키를 눌러 selection tool을 활성화 한 후, 그래프의 각 요소를 클릭해 보면 선택 외곽선이 표시된다. 일단 x, y축의 이름을 한글로 바꾸고, avg라는 범례 제목도 한글로 바꾸자.
그래프의 각 부분을 클릭하면, 우리가 선택하고자 하는 텍스트의 개별 선택이 불가능하다. 개별

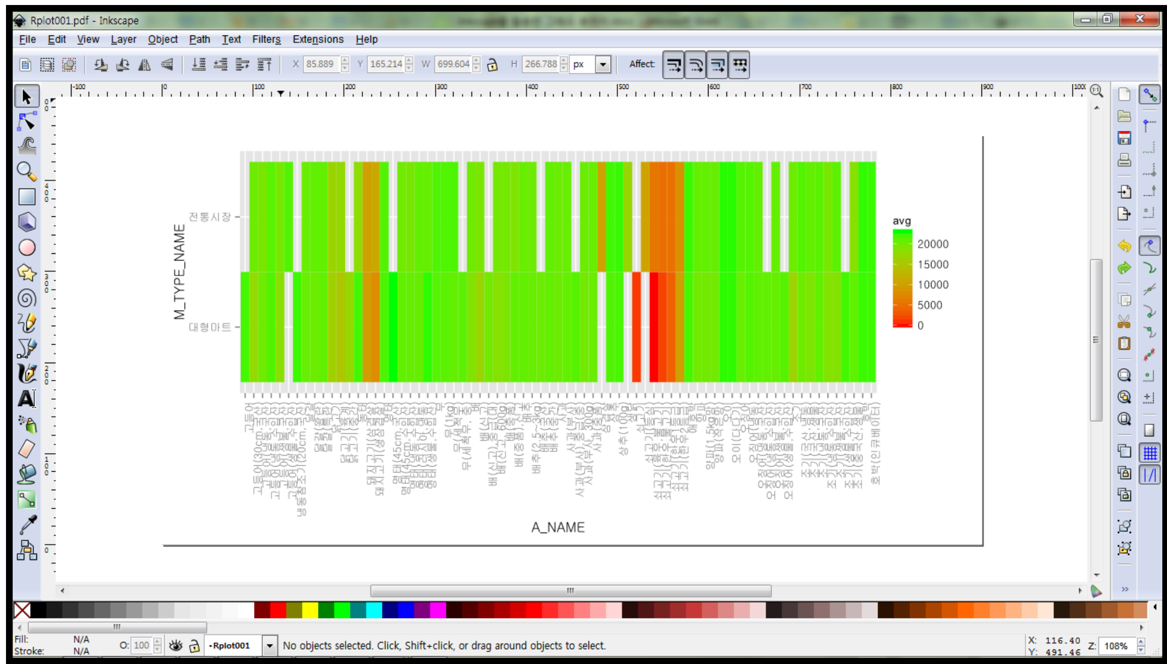

그림. 5.4: 잉크스케이프 pdf 로딩 후 화면

선택을 할 수 있도록 [Object - Ungroup] 으로 그룹해제를 한다. 그렇게 하면 각 축의 이름과 범례 제목이 분리되는 것을 볼 수 있다. 분리된 각 텍스트 중에서 x축의 텍스트를 선택하고 이를 교체해 보자. 일단 텍스트를 선택하고 [Text - Text and Font] 를 선택하면 그림5.5와 같은 폰트와 텍스트 편집 창이 뜬다.

적절한 폰트와 텍스트를 입력하고 'Apply' 버튼을 눌러 적용한다. 이렇게 우리가 편집하고자 한 세 가지 항목에 대해 편집을 하자(필자는 '다음체'를 폰트로 사용했으며 이것도 역시 인터넷에서 무료로 내려받을 수 있다).
x축의 데이터 텍스트들도 너무 커서 서로 침범하고 있으므로 이것들도 조절하자. x축 데이터

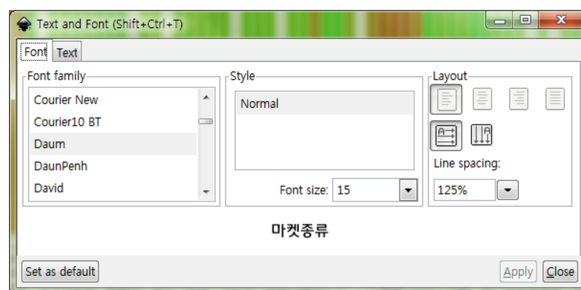

그림. 5.5: Text and Font 화면

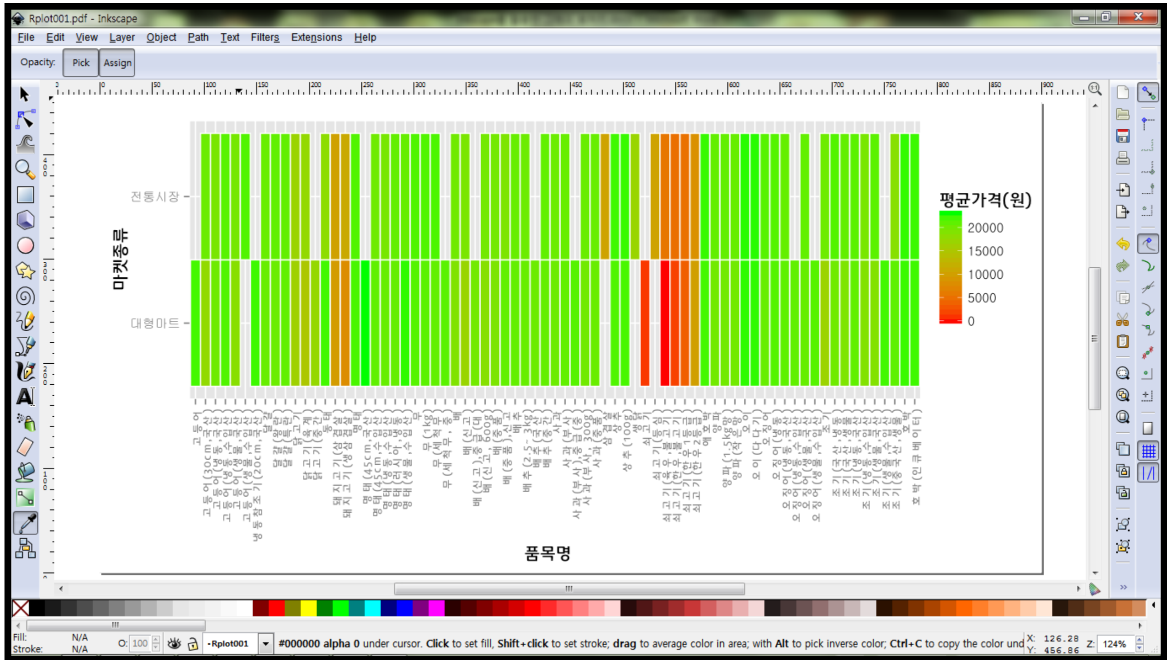

그림. 5.6: 테두리 색상을 바꾼 후의 그래프

텍스트들의 폰트 크기를 조절하면 그래프와 정렬이 흐트러지므로 커서로 적절하게 객체들을 옮겨서 열을 맞추면 된다.
대략적으로 이 과정을 거쳐 진행됐다. 하지만 왠지 ggplot2의 갈색 배경 테마가 어색하다. 각 그리드(grid)의 상자를 하얀색으로 구분하면 좀더 깔끔해 보일 것 같다.
이를 위해 범례 및 레이블을 제외한 그래프 본체를 선택하자. 왼쪽 도구 상자에서 [Pick colors from images] 툴을 선택하면 스포이드가 올라온다. 이 툴은 특정 주변색으로 객체의 배경과 테두리의 색상을 바꾸기 위한 것이다. 객체와 이 툴을 선택한 후 [shift] 키를 누르고 마우스 왼쪽 버튼을 클릭해 테두리 색상으로 쓰일 색깔을 선택해주면 그림5.6과 같이 테두리 색상이 바뀐다. 색 선택은 자유이나 바탕에 있는 하얀색으로 클릭해주면 훨씬 낫다.

훨씬 깔끔하게 플로팅된 것을 확인할 수 있다. 배경색도 흰색에 가까운 색으로 바꾸자. 이 역시 테두리 색을 바꾸는 과정과 동일하다. 다만 회색으로 바꾸기 위해서는 '채움(fill)' 명령으로 해야 된다. 채움 명령은 단순히 [Pick color from image] 툴을 선택하고 채우고 싶은 색상 위에서 왼쪽 마우스 클릭만으로 가능하다. 하지만 단순히 이런 과정을 통해 색상을 바꾼다면 모두 하얀색으로 바뀔 것이다. 원하는 모습으로 바꾸자면 내부 그래프 객체에 대해 그룹을 해제해야 된다. 이 이유는 그룹 자체가 각 그래프 객체의 오브젝트들의 레이어를 이루고 있기 때문으로, 그룹을 해제한 후 배경 레이어만 선택해 색상을 적용하기 위함이다. 모두 그룹 해제를 한 후에 앞의 과정으로 진행하면 회색 배경이 하얀색 배경으로 그림5.7과 같이 바뀐 것을 볼 수 있다.

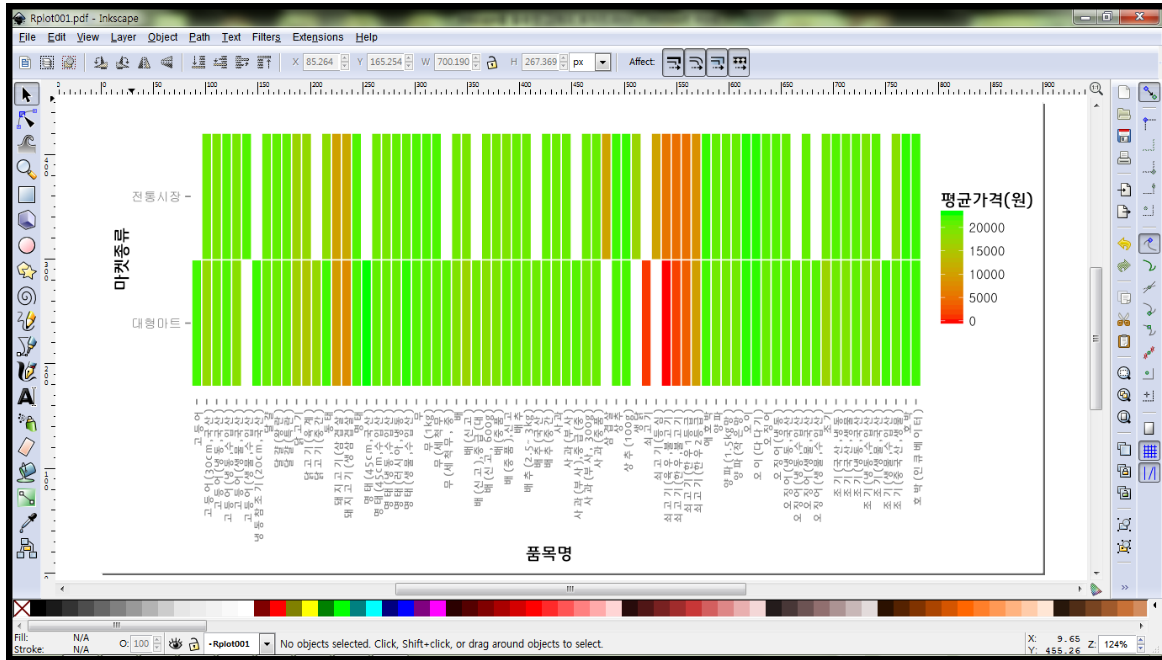

그림. 5.7: 그래프 배경 색상을 바꾼 후의 그래프

마지막으로 x축의 눈금자와 텍스트가 멀리 떨어져 부자연스러워 보인다. 이들 객체를 선택툴로 모조리 선택해 위로 이동해 주자! 마지막으로 [Create and edit text object] 툴을 선택해 적당한 곳에 그림5.8과 같이 제목을 넣어주자.

5.4 그래프 후처리와 나머지 작업

잉크스케이프의 장점은 역시 위지위그(WYSIWYG)와 무료라는 데 있다. 물론 고가의 일러스트레이터의 모든 기능을 망라하지는 못하지만, 필자와 같이 급하게 파워포인트에 쓰일 그래프를 만드는데 사용할 수 있는 수준의 유용한 툴이다.
하지만 필자는 그래프조차도 재사용성(reproducible)이 가능해야 된다고 생각하는 사람 중 한 명이다. 다른 누군가 어떤 환경에서 코드를 실행해도 같은 그래프가 나오도록 하는 건 그만큼 가치 있고 소중한 작업이기 때문이다. 특히 리서치 분야에서는 후학을 위해 꼭 그렇게 할 필요가 있다.
이를 위해서는 코드로 가능한 한 많은 부분을 커버해야 된다. 따라서 잉크스케이프 사용은 일회성의 작업 혹은 반드시 필요한 곳에 사용하는 유연성을 가져야 된다는 것을 명심했으면 한다.

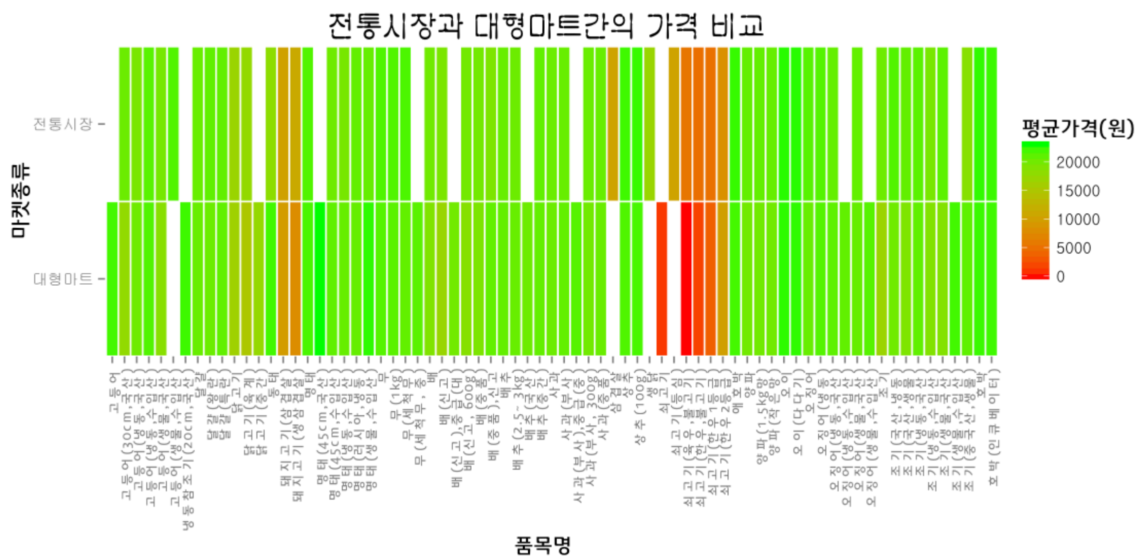

그림. 5.8: x축 텍스트 위치 조정과 제목을 추가한 그래프

Chapter 6

R로 그래프 플로팅을 하기 위한 몇 가지 팁

이번 장에서는 R로 그래프 플로팅을 하기 위한 몇 가지 팁을 소개한다.

6.1 웹으로 게시할 그래프에 J(E)PG를 사용하지 말자.

대부분 많은 사용자들이 디지털카메라를 사용하면서 JPG라는 이미지 포맷을 익숙하게 사용한다. 따라서 아무 의심없이 R에서는 그래프 플로팅을 파일로 뽑아낼 때 JPG를 사용할 수 있는데, 이는 JPG라는 포맷의 성격을 잘못 알고 사용하는 것이다. JPEG포맷은 이미지나, 색상의 경계들에 대해서 양자화 압축을 수행하는데, 이 부분 때문에 그래프에서 라인과 같은 성분의 경우 흐릿하게 보일 수 있기 때문이다.
아래는 JPG와 PNG를 동일 크기의 이미지로 출력하기 위한 예제 코드이다.

```
> library(ggplot2)
>
> jpeg(width=500, height=500,filename="jpg.jpg")
> ggplot(diamonds, aes(carat, price)) + geom_line(aes(colour=clarity))
> dev.off()
>
> png(width=500, height=500,filename="png.png")
> ggplot(diamonds, aes(carat, price)) + geom_line(aes(colour=clarity))
> dev.off()
>
> cairo_pdf(filename="pdf.pdf")
> ggplot(diamonds, aes(carat, price)) + geom_line(aes(colour=clarity))
> dev.off()
```

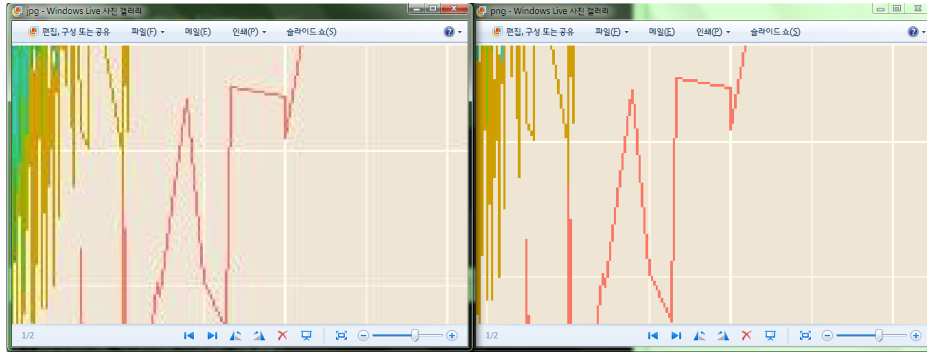

그림. 6.1: JPG와 PNG 그래프 출력 결과 비교

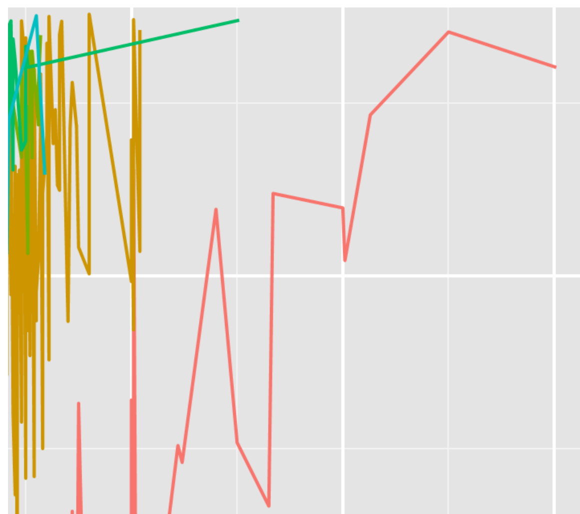

그림. 6.2: PDF 출력 확대 화면

그림6.1은 실제 그래프를 뽑아본 뒤 이를 같은 배율로 확대한 화면을 캡처한 화면이다(왼쪽이 JPG이고 오른쪽이 PNG이다).

JPG 파일에서 경계부분의 색상이 이상하며, 라인 색상 역시 바랜듯한 빛을 띠고 있다. 바로 이런 이유 때문에 그래프 출력에 JPG를 사용하지 말라는 것이다. 그리고 JPG, PNG 모두 계단 현상이 있는 것을 볼 수 있을 것이다. 이는 레스터 이미지(Raster Graphics)의 표현방식 때문이다. 이런 계단현상은 이미지를 확대할 경우 더욱 부각되는 단점이 있다. 따라서 PNG에 만족하지 못한다면, PDF방식의 이미지 출력을 사용할 것을 추천한다. PDF로 출력할 경우 같은 그래프가 그림6.2 방식으로 표현된다.

이 경우에는 아무래 확대해도 계단현상이 일어나지 않는다.

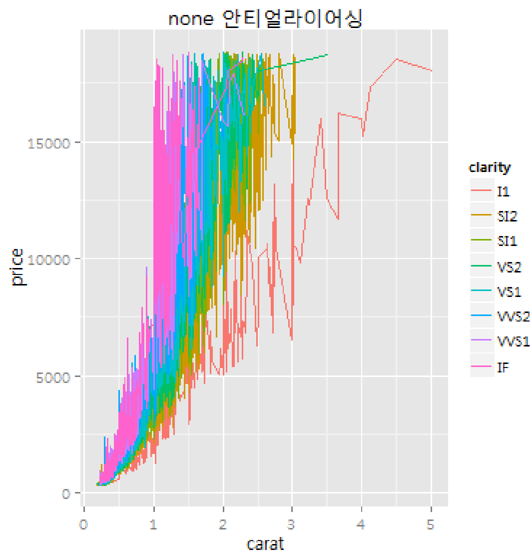

그림. 6.3: 안티얼라이어싱 사용 안함

6.2 anti-aliasing을 활성화 하라.

대부분의 시스템은 anti-aliasing을 활성화 해놓지만, 몇몇 시스템의 경우 이 부분을 확인해야 될 경우가 있다. 이 기술은 경사방향의 선분이나 경계선에서 생기는 계단형의 모양을 제거하는 기술로서 픽셀로 표현된 기울어진 선분이 자연스럽게 연결되게 한다.
아래의 첫 번째 그래프가 기본 옵션으로 플로팅한 결과이며 두 번째는 cairo 라이브러리를 활용한 subpixel anti-aliasing을 준 옵션이다. 물론 둘 다 PNG로 뽑은 결과인데, 두 번째 그래프가 선분을 표현한 부분에서 좀더 자연스러운 이미지를 제공하는 것을 볼 수 있다.

최근의 R 버전에는 cairo가 기본 탑재가 되어 있어서 아래와 같은 명령으로 위와 같은 anti-aliasing기능을 사용할 수 있다.

```
> #윈도우 맑은 고딕 폰트를 R 폰트시스템에 연동
> windowsFonts(malgun=windowsFont("맑은 고딕"))
>
> png(width=400, height=400,filename="pngnaa.jpg", family="malgun")
> ggplot(diamonds, aes(carat, price), family="malgun") + geom_line(aes(colour=clarity)) + ggtitle("none 안티얼라이어싱")
> dev.off()
>
> png(width=400, height=400,filename="pngaa.jpg", type="cairo",antialias="subpixel", family="malgun")
> ggplot(diamonds, aes(carat, price), family="malgun") + geom_line(aes(colour=clarity)) + ggtitle("안티얼라이어싱")
> dev.off()
```

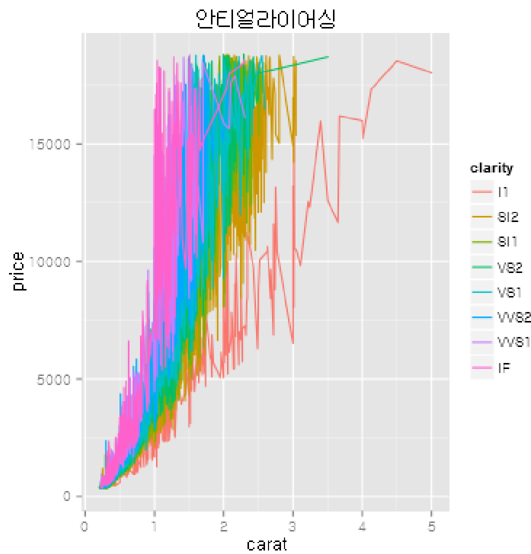

그림. 6.4: 안티얼라이어싱 사용

6.3 정확한 디바이스 드라이버를 사용해 그래프를 저장하라.

윈도우에서 R 기본 GUI를 사용하거나, RStudio를 사용하거나 했을 때, 기본적인 그래프를 저장할 옵션이 존재한다. 사실 그런 옵션을 사용해서 저장하는 건 추천하지 않는다. 왜냐면 GUI 자체가 상세한 옵션을 제공해 주지 않기 때문이다. 따라서 GUI에서 제공하는 출력은 그래프가 의도하는 바대로 데이터를 잘 반영하고 있는지 확인하는 용도로 사용하고, 이를 가지고 웹에 게시하거나 출력물을 만드는 행위는 지양하기 바란다.
그렇다면 어떤 방식으로 출력물을 만드는지 궁금하다 할 수 있겠는데, 1, 2번 항목에서 예제코드를 출력한 방식으로 출력 파일을 생성하면 된다.
예를 들자면 아래와 같다.

```
> png(…)    # jpeg(..), cairo_pdf(…), tiff(…)
> ….. #출력하고자 하는 그래프 함수들을 실행한다.
> dev.off()
```

6.4 필요시, 고해상도 이미지로 출력하라.

R에서 그래프는 항상 출력물 기준으로 표현된다. 그리고 출력물의 기본 사이즈는 인치(inch)로 표현되는데, 이 때문에 png()나 jpeg()와 같은 명령어들에서 res라는 파라메터가 중요하게 된다.

res에 들어가는 숫자의 단위는 pixel per inch로서 1인치를 몇 픽셀로 표현할 것인지를 결정하게 되는 옵션이다. 이 res는 또한 이 강좌에서는 dpi(dot per inch)하고 구분하지 않겠다. 도트는 프린터 출력의 표현 기준이 되며 픽셀은 모니터 출력의 기준이 되기 때문인데 어차피 같은 이미지를 무엇으로 표현하느냐의 문제니 구분을 두지 않겠다는 이야기다.
여기서 가정을 하자면, 5 * 5 인치 크기의 이미지를 파워포인트 슬라이드에 넣고자 하며 220 ppi가 지원되는 파워포인트 2010 슬라이드에 첨부하고자 한다[1].
R의 모든 플로팅은 72ppi로 맞춰져 있다. 따라서 5인치를 표현하기 위해서는 가로, 세로 360(5 * 72) 픽셀로 정해져야 된다. 따라서 아래와 같은 그래프 생성 코드를 만들 수 있다.

```
> png(width=360, height=360,filename="pngaa360.jpg",
> type="cairo",antialias="subpixel", family="malgun") # res = 72 옵션은 생략
> ggplot(diamonds, aes(carat, price), family="malgun") +
>   geom_line(aes(colour=clarity))  + ggtitle("72 ppi(dpi)")
> dev.off()
```

그리고 220ppi를 지원하는 같은 내용의 이미지를 만들기 위한 코드를 아래와 같은데, 이 이미지는 가로, 세로 1100(220 * 5) 픽셀로 계산된다.

```
> png(width=1100, height=1100,filename="pngaa1100.jpg",
> type="cairo",antialias="subpixel", family="malgun", res=220)
> ggplot(diamonds, aes(carat, price), family="malgun") +
>   geom_line(aes(colour=clarity))  + ggtitle("220 ppi(dpi)")
> dev.off()
```

실제 모니터에서 해상도의 차이를 볼 수 있는 방법은 이 두 파일을 파워포인트나 pdf를 통해 보는 것이다.
그림6.5는 이 두 파일을 파워포인트 슬라이드쇼를 통해 출력한 결과이다.
사실 이렇게 봐서는 차이가 크게 나지 않을 것이다. 하지만 이 문서를 pdf로 출력 후 확대해서 보면 그림6.6과 같은 차이가 있다.
만일 220 dpi가 지원되는 빔 프로젝터 기기가 있다면 그래프 화면을 보는 청중은 그래프 퀄리티 차이를 심하게 느낄 가능성이 있는 것이다.
게다가 일반적으로 책을 뽑아내는 프린터의 dpi는 600정도 지원하는데, 이를 고려하지 않고 그래프 이미지를 생성하지 않는다면 독자는 뭉개진 그래프를 볼 가능성이 많다.

6.5 출력을 위해서라면 PDF를 활용하라

사실 DPI에 대한 논의는 픽셀로 표현되는 레스터 이미지를 사용할 경우에 해당된다. 이미지를 PDF로 바로 출력하면 이런 고민은 사라지게 된다. 또한 벡터 이미지가 필요하다면 PDF가

[1] 일반적으로 220을 지원한다.

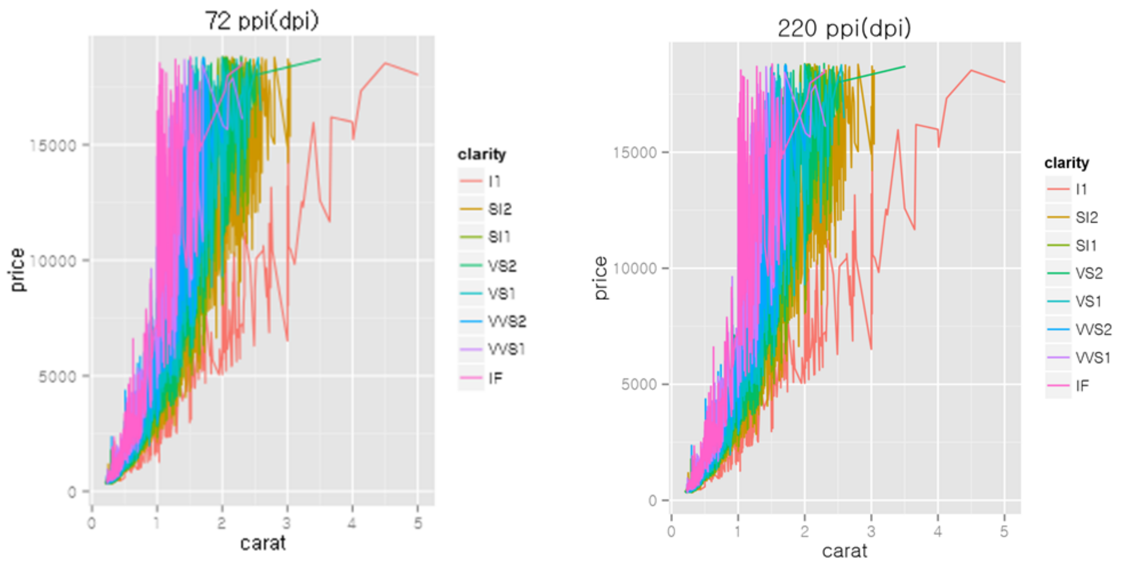

그림. 6.5: DPI에 따른 차이

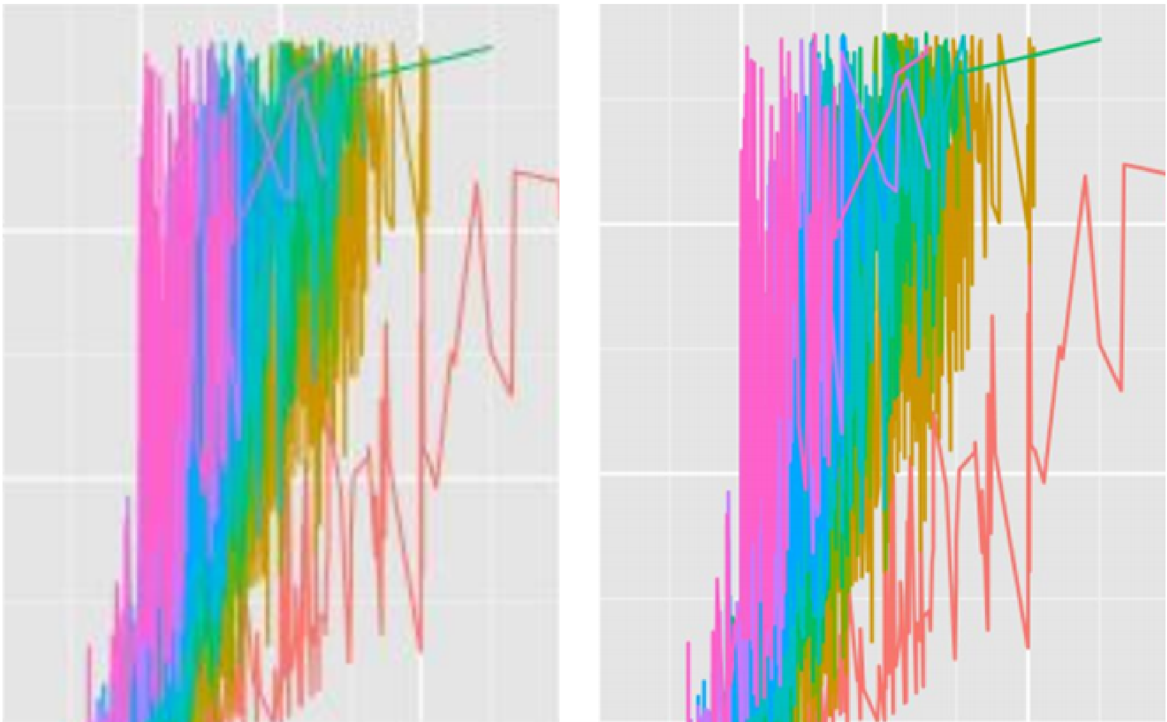

그림. 6.6: DPI에 따른 차이 (자세하게)

최적의 방법이다. 물론 R에서 SVG같은 벡터 포맷도 지원을 하고 있으나 한글 문제가 있어서 우리나라에서 사용상의 애로사항이 있다. 그리고 pdf() 명령을 통한 디바이스 출력을 사용하는 건 좋은 방법이나 이 또한 한글을 표현하는데 문제가 있으니 cairo_pdf() 명령을 사용하길 추천한다.
PDF는 출력시 이미지의 고품질을 보장하며, 대부분의 PC에서 뷰어를 가지고 있어서 배포에도 용이한 장점이 있다.

참고 문헌

Allaire J, Horner J, Marti V, Porte N (2013). *markdown: Markdown rendering for R.* R package version 0.5.4, URL `http://CRAN.R-project.org/package=markdown`.

Eddelbuettel D, François R (2011). "Rcpp: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. URL `http://www.jstatsoft.org/v40/i08/`.

Fox J (2005). "The R Commander: A Basic Statistics Graphical User Interface to R." *Journal of Statistical Software*, **14**(9), 1–42. URL `http://www.jstatsoft.org/v14/i09`.

Graves S, Dorai-Raj S, Francois R (2012). *sos: sos.* R package version 1.3-5, URL `http://CRAN.R-project.org/package=sos`.

Grothendieck G (2012). *sqldf: Perform SQL Selects on R Data Frames.* R package version 0.4-6.4, URL `http://CRAN.R-project.org/package=sqldf`.

Hahsler M, Buchta C, Gruen B, Hornik K (2012). *arules: Mining Association Rules and Frequent Itemsets.* R package version 1.0-12., URL `http://CRAN.R-project.org/`.

Jeon H (2012). *KoNLP: Korean NLP Package.* R package version 0.76.8, URL `http://CRAN.R-project.org/package=KoNLP`.

Leisch F (2002). "Sweave: Dynamic Generation of Statistical Reports Using Literate Data Analysis." In W Härdle, B Rönz (eds.), *Compstat 2002 — Proceedings in Computational Statistics*, pp. 575–580. Physica Verlag, Heidelberg. ISBN 3-7908-1517-9, URL `http://www.stat.uni-muenchen.de/~leisch/Sweave`.

R Core Team (2012). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL `http://www.R-project.org/`.

RStudio Team (2012). *RStudio: Integrated Development Environment for R.* RStudio, Inc., Boston, MA. URL http://www.rstudio.com/.

Wickham H (2009). *ggplot2: elegant graphics for data analysis.* Springer New York. ISBN 978-0-387-98140-6. URL http://had.co.nz/ggplot2/book.

Wickham H (2011). "The Split-Apply-Combine Strategy for Data Analysis." *Journal of Statistical Software*, **40**(1), 1–29. URL http://www.jstatsoft.org/v40/i01/.

Wickham H (Submitted). "Tidy data." *The American Statistician.*

Williams GJ (2011). *Data Mining with Rattle and R: The art of excavating data for knowledge discovery.* Use R! Springer. URL http://www.amazon.com/gp/product/1441998896/ref=as_li_qf_sp_asin_tl?ie=UTF8&tag=togaware-20&linkCode=as2&camp=217145&creative=399373&creativeASIN=1441998896.

Xie Y (2013). *knitr: A general-purpose package for dynamic report generation in R.* R package version 1.0.5, URL http://CRAN.R-project.org/package=knitr.