# 데이터 취득과 데이터 가공: SQL과 dplyr

# 데이터 취득과 데이터 가공이란 무엇이며, 왜 중요한가?

"데이터 과학자는 쓸모있는 통계학자이다" (해들리 위컴)

"데이터 과학의 기본 기술은 데이터 를 여러 소스로부터 분석 플랫폼으로 읽어 들이는 데이터 취득(data acquisition, data import)과 데이터를 분석 플랫폼 안에서 필요한 모양으로 가공해내는 데이터 가공(data processing, data wrangling) 기술이다. 이 중 데이터 취득은 요리 재료를 얻어내는, 즉 장을 보는 기술이다. 데이 터 가공은 칼질에 가깝다."

"데이터 취득 및 가공 능력은 통계학자를 진정한 데이터 과학자로 만들어준다."

# 데이터취득

# 예제 데이터 찾기

1. UCI 머신러닝 리포 [UCI Machine Learning Repository] https://archive.ics.uci.edu/ml/index.php
2. R에서 제공하는 예제 데이터.
    a. help(package='datasets')
3. 머신러닝/데이터 과학 공유/경연 사이트 캐글 (https://www.kaggle.com/)
4. 위키피디아의 머신러닝 연구를 위한 데이터세트 리스트 (https://goo.gl/SpCOlK )

# 표 형태 텍스트 파일

- 대부분 컴마나 탭문자로 열이 구분된 형태이다.
- 옛 방법: read.table 과 그 친구들
- 새 방법: read_csv (tidyverse 스타일)
- 두 방법의 차이는?
- factor 되는 것을 조심하자

# 보스턴 주택 데이터 다운받고 R로 읽어들이기

- https://archive.ics.uci.edu/ml/datasets/Housing
- Unix에서 : curl 로 파일 다운로드
- R에서 read.table() read_csv, 등으로 읽어 들이기.
- boston 이란 data.frame (혹은 tibble) 으로 저장.
- 변수명을 의미있는 이름으로 바꾸기 names(boston) <- ...
- glimpse() 로 자료를 살펴보자

변수설명 파일을 통해 변수는 다음 의미를 가지고 있음을 알 수 있다.

- crim: 범죄발생률
- zn: 주거지 중 25000 $ft^2$ 이상 크기의 대형주택이 차지하는 비율
- indus: 소매상 이외의 상업지구의 면적 비율
- chas: 찰스 강과 접한 지역은 1, 아니면 0인 더미 변수
- nox: 산화질소 오염도
- rm: 주거지당 평균 방 개수
- age: 소유자 주거지(비 전세/월세) 중 1940년 이전에 지어진 집들의 비율
- dis: 보스턴의 5대 고용중심으로부터의 가중평균거리
- rad: 도시 순환 고속도로에의 접근 용이 지수
- tax: 만 달러당 주택 재산세율
- ptratio: 학생-선생 비율
- black: 흑인 인구 비율(Bk)이 0.63과 다른 정도의 제곱, $1000(Bk - 0.63)^2$
- lstat: 저소득 주민들의 비율 퍼센트
- medv: 소유자 주거지(비 전세/월세) 주택 가격

# 아주 큰 외부파일

R에서 data.table 패키지 활용

Unix 에서 head 명령을 사용해 처음 10K 줄을 잘라내는 것도 좋다

R 함수 안에 max.lines= 등을 활용하여 처음 몇줄만 읽어내기

"너무 일찍 최적화하지 말자"

# 엑셀 파일 읽어 들이기

방법 1: 엑셀에서 csv 로 저장 후 R로 읽어 들이기.

방법 2: 적절한 패키지 사용 (readxl)

# RDBMS + SQL

"거의 모든 데이터 과학자는 언젠가는 SQL을 사용하게 된다. 많은 회사들이 데이터를 SQL을 사용하는 RDBMS에 저장하기 때문이다. 워낙 많은 분석가가 SQL에 익숙하므로 페이스북 등 에 쓰이는 빅데이터를 위한 분산시스템인 하둡의 파일시스템에 저장된 데이터도 SQL 문법을 사용하여 처리하고 추출할 수 있는 하이브(Hive, https://hive.apache.org/)가 사용된다[Apache Hive (2016)]."

SQL 연습하기:

http://sqlzoo.net/wiki/Self_join

# R에서 SQL 연습하기

```
# install.packages("sqldf")
library(sqldf)
sqldf("select * from iris")
sqldf("select count(*) from iris")
sqldf("select Species, count(*), avg(`Sepal.Length`)
       from iris
       group by `Species`")
sqldf("select Species, `Sepal.Length`, `Sepal.Width`
       from iris
       where `Sepal.Length` < 4.5
       order by `Sepal.Width`")
```

# RDBMS에서 R로 데이터 읽어 들이기

방법1. RDBMS 에서 CSV 로 저장 후 R로 읽어들이기.

방법 2. R의 RDBMS 접속 라이브러리 사용: RPostgreSQL, RMySQL, ROracle, RODBC, ...

# 다른 소프트웨어 데이터 포맷 읽어 들이기

SAS, SPSS, ...

방법1. 타 소프트웨어세서 에서 CSV 로 저장 후 R로 읽어들이기.
방법 2. R의 foreign 패키지 사용

# 데이터 출력

CSV로 저장하기: tidyverse 패키지의 write_csv() 로 충분

# 데이터 가공

"보통 데이터 과학자의 데이터 분석 작업 시간의 70~80% 이상은 데이터 가공에 소요된다!"

데이터 가공이란, 원데이터를 여러 연산을 통해서 필요한 시각화, 모형화에서 사용할 수 있는 데이터, 즉 적절한 관측치(observations)는 행(rows)으로, 적절한 변수(variables)는 열(columns)로 되어 있는 테이블 형태로 변환하는 작업이다. 즉, 목적 데이터는

- 테이블 형태다.
- 각 행은 적절한 관측치를 나타낸다.
- 각 열은 적절한 변수를 나타낸다.

| | 변수 1 | 변수 2 | ... | 변수 p |
|---|---|---|---|---|
| 관측치 1 | | | | |
| 관측치 2 | | | | |
| ... | | | | |
| 관측치 n | | | | |

# 데이터 가공을 위한 도구

SQL

유닉스 셸

파이썬

R

# 데이터를 가공하는 전통적 방법

베이스 R은 데이터 가공을 위한 강력한 기능을 제공한다.

```
# 데이터를 로드한다.
# gapminder 패키지를 설치한다. 한 번만 실행하면 된다.
install.packages("gapminder")

# 행과 열 선택
gapminder[gapminder$country=='Korea, Rep.', c('pop', 'gdpPercap')]

# 행 선택
gapminder[gapminder$country=='Korea, Rep.', ]
gapminder[gapminder$year==2007, ]
gapminder[gapminder$country=='Korea, Rep.' & gapminder$year==2007, ]
gapminder[1:10,]
head(gapminder, 10)

# 정렬
gapminder[order(gapminder$year, gapminder$country),]

# 변수 선택
gapminder[, c('pop', 'gdpPercap')]
gapminder[, 1:3]
```

# 데이터를 가공하는 전통적 방법 (cont'd)

```
# 변수명 바꾸기: gdpPercap를 gdp_per_cap 으로 변경
f2 = gapminder
names(f2)
names(f2)[6] = 'gdp_per_cap'

# 변수 변환과 변수 생성
f2 = gapminder
f2$total_gdp = f2$pop * f2$gdpPercap

# 요약 통계량 계산
median(gapminder$gdpPercap)
apply(gapminder[,4:6], 2, mean)
summary(gapminder)
```

# R의 dplyr 패키지

베이스 R보다는 dplyr 패키지를 가능한 한 많이 사용할 것을 강력하게 추천 dplyr(“디플라이어”)는 데이터를 빨리 쉽게 가공할 수 있도록 도 와주는 R패키지다

1. 코드를 읽기 쉽다. 체인(chain) 연산자 ‘%>%’ 덕분이다.

2. 코드를 쓰기 쉽다. ‘동사(verb)’의 개수가 적고, 문법이 간단하기 때문이다. 아이디어를 코드로 옮기기 수월하다.

3. R 스튜디오를 사용한다면 변수명이 자동완성된다. 코딩이 빨라진다.

4. 데이터 프레임만 처리한다. 베이스 R의 연산자들은 데이터 프레임뿐 아니라 벡터, 행렬, 다차원 배열, 리스트에 적용된다.

5. ‘문법’과 접근 방법이 SQL과 비슷하다. SQL에 익숙한 사람은 더 쉽게 배울 수 있다.

# dplyr 의 동사(verbs)

- filter(df, 조건) (and slice( )): 행 선택
- arrange(df, 변수1, 변수2, ...): 행 정렬
- select(df, 변수1, 변수2, ...): 변수/열 선택
- mutate(df, 타겟변수1=변환, ... ): 변수 변환
- summarize(df, 타겟변수1=통계함수, ...): 변수 요약
- distinct( )
- sample_n( ) and sample_frac( )

# 행을 선택하는 filter( )

```
gapminder %>% filter(country=='Korea, Rep.')
gapminder %>% filter(year==2007)
gapminder %>% filter(country=='Korea, Rep.' & year==2007)
```

# 행(관측치)을 정렬하는 arrange( )

```
arrange(gapminder, year, country)
gapminder %>% arrange(year, country)
```

# 열(변수)을 선택하는 select( )

```
select(gapminder, pop, gdpPercap)
gapminder %>% select(pop, gdpPercap)
```

# 변수를 변환하는 mutate( )

```
gapminder %>%
  mutate(total_gdp = pop * gdpPercap,
         le_gdp_ratio = lifeExp / gdpPercap,
         lgrk = le_gdp_ratio * 100)
```

# 요약 통계량을 계산하는 summarize( )

```
gapminder %>%
  summarize(n_obs = n( ),
            n_countries = n_distinct(country),
            n_years = n_distinct(year),
            med_gdpc = median(gdpPercap),
            max_gdppc = max(gdpPercap))
```

# 기타 동사

랜덤 샘플을 위한 sample_n( )과 sample_frac( )

고유한 행을 찾아내는 distinct( )

# group_by를 이용한 그룹 연산

```
gapminder %>%
  filter(year == 2007) %>%
  group_by(continent) %>%
  summarize(median(lifeExp))
```

# dplyr 명령의 공통점, 함수형 프로그래밍, 체이닝

```
d1 = filter(gapminder, year == 2007)
d2 = group_by(d1, continent)
d3 = summarize(d2, lifeExp = median(lifeExp))
arrange(d3, -lifeExp)

arrange(
  summarize(
    group_by(
      filter(gapminder, year==2007), continent
    ), lifeExp=median(lifeExp)
  ), -lifeExp
)
```

**VS**

```
gapminder %>%
  filter(year == 2007) %>%
  group_by(continent) %>%
  summarize(lifeExp = median(lifeExp)) %>%
  arrange(-lifeExp)
```

# dplyr에서 테이블을 결합하는 조인 연산자

SQL과 유사

inner_join

left_join

right_join

full_join

intersect

union

# SQL과 dplyr

| 데이터 처리 작업 | R | SQL |
|---|---|---|
| 1. 행 선택, filter | `df %>% filter(x>0)` | `SELECT *`<br>`FROM df`<br>`WHERE x > 0` |
| 2. 정렬, arrange | `df %>% arrange(x)` | `SELECT * FROM df`<br>`ORDER BY x` |
| 3. 변수 선택 | `df %>% select(x)` | `SELECT x FROM df` |
| 4. 변수 변환 | `df %>% mutate(y=f(x))` | `SELECT f(x) AS y FROM df` |
| 5. 요약 통계량 계산 | `df %>% summarize(avg_x=mean(x))` | `SELECT avg(x) AS avg_x`<br>`FROM df` |
| 6. 랜덤 샘플링 | `df %>% sample_n(100)`<br>`df %>% sample_frac(0.1)` | 없음* |
| 7. 유일값 계산 | `df %>% select(x) %>%`<br>`distinct()` | `SELECT DISTINCT(x)`<br>`FROM df` |
| 8. 그룹핑 | `df %>%`<br>`group_by(x) %>%`<br>`summarize(total=n())` | `SELECT x, count(*) AS total`<br>`FROM df`<br>`GROUP BY x` |
| 9. 이너 조인(inner join) | `inner_join(x, y, by="a")` | `SELECT *`<br>`FROM x JOIN y`<br>`ON x.a = y.a` |
| 10. 레프트 조인(left join) | `left_join(x, y, by="a")` | `SELECT *`<br>`FROM x LEFT JOIN y`<br>`ON x.a = y.a` |
| 11. 풀 조인(full join) | `full_join(x, y, by="a")` | `SELECT *`<br>`FROM x FULL JOIN y`<br>`ON x.a = y.a` |
| 12. 합집합(union) | `union(x, y)`<br>`union_all(x, y)` | `SELECT * FROM x`<br>`UNION`<br>`SELECT * FROM y` |

* HIve SQL에는 제공된다(https://goo.gl/q2mISm 참고).

# 요약

데이터 취득과 가공 능력의 중요성

예제 데이터 찾기

테이블형 데이터 취득 read.table / read_csv

SQL의 중요성

R의 dplyr 을 이용한 자료 가공