R tidyverse 마스터 클래스

3강 - `dplyr` 패키지와 데이터 베이스

슬기로운통계생활

Issac Lee

Big 데이터에 대처하는 우리들의 자세

데이터가 큰게 뭐가 문제?

한정된 메모리

RAM (Random Access Memory)

- 컴퓨터의 두뇌 용량 (단기 메모리)
- 겁나게 빠름, but 컴터가 꺼지면 사라짐.
- 요즘 컴퓨터 16GB 정도 딸려나오쥬?
- 브라우져 켜고, RStudio 켜고 그려면 RAM을 이용하는 것.
- R에서 데이터를 불러온다는 의미는 RAM에 저장을 한다는 의미.

한정된 RAM, 큰 데이터

2GB 데이터

- 요즘 2GB가 큰 편이 아님.
- 연산에 따라 작은 데이터도 얼마든지 큰 메모리를 잡아먹게 할 수 있음
- 즉, 어느정도 데이터가 커지면, 그 **데이터를 다루는데 들어가는 메모리는 더 커야함.**

```
x <- 1:3
y <- -c(1:3)
mygrid <- expand.grid(x, y)
object.size(mygrid)
```

```
## 2320 bytes
```

해결책? R $\leftrightarrow$ DB

데이터 베이스를 만들자.

- 데이터 저장소 (DB)
- 데이터 저장소에 접근해서 다룰 수 있도록 해주는 관리 시스템 (DBMS)
- DBMS에서 사용하는 언어: SQL (시퀄이라고 읽음.)

R과 궁합 좋은 DBMS

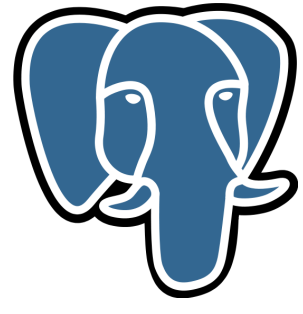
- 모든 DB 시스템에 대한 패키지가 존재함. 하지만 그 중 더 궁합이 좋은 것이 존재

유명 프로그램들

- SQLite
- MySQL, MariaDB
- PostgreSQL
- BigQuery

- 개인, 중소 사업자에게 좋음

- 기업들이 이용



SQLite

MS DB제품 시연용

장점

- 처음 시작하는 사람들이 사용하기 좋음
- 세상 가벼움 파일 3개??? 실화?
- 모든 시스템에서 돌아감.
- 로컬에서 사용하기 딱 좋음.

단점

- 기본 기능 밖에 없다.
- 보안 기능 없다. (이메일, 주민번호저장..?)

SQLite 설치

코드 한 줄

```
install.packages("RSQLite")
```

- `RSQLite` 패키지 안에 SQLite 설치가 포함 되어있음.
- 얼마나 작으면 그럴까? 1.82MB
- 자신의 시스템에 맞게 `sqlite-tools` 압축파일을 받고 풀어봅시다.
  - 환경변수 등록
  - `sqlite3.exe` 실행

SQLite 다루기

기초 테이블 만들기

- `.db` 확장자를 사용함
- `csv` 파일로부터 테이블 만들기

```
.open 'moving_data.db'
.mode csv
.headers on
.import 'file_name.csv' table_name
```

- `testdata.csv` 파일을 불러와 `test_tbl`이라는 테이블에 저장해보자.

```
SELECT * FROM test_tbl LIMIT 5;
```

SQLite 다루기 - 테이블 생성

테이블 만들기

- CREATE TABLE

```
CREATE TABLE my_score (
  "id" INT,
  "name" TEXT,
  "score" INT
);
```

table 리스트 확인

- `.tables` 명령어로 현재 저장된 테이블 확인

테이블에 값넣기

- INSERT INTO

```
INSERT INTO my_score (id, name,
VALUES (123, "student 1", 10)
```

테이블 지우기

- DROP TABLE

```
DROP TABLE my_score;
```

- `.db` 파일의 용량에 반영하기 위해서는 `vacuum;` 실행

지금이 SQL 시간이냐!!

`RSQLite` **패키지를 이용해봅시다.**

- `dbConnect()`: db 파일과 연결 시켜줌.
- `dbDisconnect()`: db 파일과 연결 시켜줌.
- `unlink("data.db")`: 특정 db 파일과 연결 해제

```
library(DBI)
con <- dbConnect(RSQLite::SQLite(), "./data/test.db")
dbListTables(con)
```

```
## [1] "moving_data"    "myfirst_table"  "reference_data" "sqlite_sta
## [5] "sqlite_stat4"   "test_tbl"
```

db에 테이블 만들어 넣기

R 데이터 프레임 to DB 테이블

문법

- `dbWriteTable(con, "tbl_name", dataframe)`
- `dbGetQuery(con, "SQL 문법")`

```
## [1] "moving_data"    "myfirst
## [5] "sqlite_stat4"   "test_tb
```

```
dbGetQuery(con, "SELECT * FROM m
```

```
dbWriteTable(con, "myfirst_table
dbListTables(con)
```

```
##    mpg cyl disp  hp drat    w
## 1 21.0   6  160 110 3.90 2.62
## 2 21.0   6  160 110 3.90 2.87
## 3 22.8   4  108  93 3.85 2.32
```

잠깐 이거 tidyverse
마스터 클래스 아니냐고..

`dbplyr` 패키지

dplyr의 백앤드 패키지

```
install.packages("dbplyr")
```

- 너는 dplyr만(이라도??) 잘해.. 나머진 내가 알아서 할게..
- `dplyr` 패키지 안의 함수에 DB table이 들어오면 자동으로 그 함수들에 대응하는 `dbplyr` 함수들이 실행됨.
- `dplyr` 사용자 입장에서는 SQL 언어가 어떻게 돌아가는지 몰라도 됨.

DB 테이블 불러오기

DB 테이블 to `dplyr`

```
mtcars_db <- tbl(con, "myfirst_t
mtcars_db %>%
  select(mpg:hp) %>%
  filter(mpg > 20)
```

- 일반 데이터 프레임과 다른 점
- `lazy` evaluation:
  - dplyr 코드는 SQL로 번역
  - SQL 코드는 db로 보내짐

```
## # Source:   lazy query [?? x
## # Database: sqlite 3.36.0
## #   [C:\Users\issac\Documents
##       mpg   cyl  disp    hp
##     <dbl> <dbl> <dbl> <dbl>
##  1  21        6 160     110
##  2  21        6 160     110
##  3  22.8      4 108      93
##  4  21.4      6 258     110
##  5  24.4      4 147.     62
##  6  22.8      4 141.     95
##  7  32.4      4  78.7    66
##  8  30.4      4  75.7    52
##  9  33.9      4  71.1    65
## 10  21.5      4 120.     97
## # ... with more rows
```

무엇이 게으르다는 것일까?

- `summary_db`가 작성 되어도 아무것도 일어나지 않음.

```
summary_db <- mtcars_db %>%
  group_by(gear) %>%
  filter(mpg > 17) %>%
  tally()
```

- 실제 코드가 돌아가는 순간은 실행할 때 돌아감.

```
summary_db
```

```
## # Source:   lazy query [?? x 2]
## # Database: sqlite 3.36.0
## #   [C:\Users\issac\Documents\Teaching\R-playbook\r-for-tidyverse\
##    gear     n
```

`dplyr`로 SQL 공부하기?

내가 짠 dplyr 코드를 SQL로

- `show_query()`: database가 연결된 `dplyr()` 객체를

```
summary_db %>%
  show_query()
```

```
## <SQL>
## SELECT `gear`, COUNT(*) AS `n`
## FROM `myfirst_table`
## WHERE (`mpg` > 17.0)
## GROUP BY `gear`
```

dplyr 데이터 프레임으로 바꿔오기

DB to tibble

- `collect()`: 데이터 베이스 객체가 연결된 dplyr 코드의 결과값을 `tidyverse`에서 사용하는 데이터 프레임 객체인 `tibble`로 바꿔줌.

```
summary_tibble <- summary_db %>%
  collect()
summary_tibble
```

```
## # A tibble: 3 x 2
##    gear     n
##   <dbl> <int>
## 1     3     6
## 2     4    12
## 3     5     3
```

moving_data DB를 만들자

```
moving_data <- read_csv("./data/seoul_moving_202107_09_hr.csv")
reference_data <- readxl::read_excel("./data/reference.xlsx")
names(moving_data) <- gsub(" ", "", names(moving_data))
names(moving_data)[9:10] <- c("평균이동시간_분", "이동인구_합")
names(reference_data) <- c("시도코드", "시군구코드", "시군구이름", "전체이름")
```

- moving_db 생성

```
copy_to(con, moving_data, "moving_data",
        temporary = FALSE,
        indexes = list(
            "대상연월", "요일", "도착시간",
            "출발시군구코드", "도착시군구코드")
        overwrite = TRUE)
```

- reference_db 생성

```
copy_to(con, reference_data,
        "reference_data",
        temporary = FALSE,
        indexes = list("시군구코드"),
        overwrite = TRUE)
```

평균 이동 시간을 이용한 여행타입 분류

```
moving_db <- tbl(con, "moving_data")
reference_db <- tbl(con, "reference_data")

moving_db %<>%
    mutate(평균이동시간_시 = 평균이동시간_분 / 60) %>%
    mutate(여행타입 =
               case_when(
                   between(평균이동시간_시, 0, 0.5) ~ "단기",
                   between(평균이동시간_시, 0.5, 1) ~ "중기",
                   평균이동시간_시 >= 1 ~ "장기",
                   TRUE ~ as.character(평균이동시간_시)
               )) %>%
    relocate(여행타입)
```

서울에서 가장 인구 이동이 많은 지역구은 어디인가?

- `summarize()`: 내가 원하는 통계치를 그룹별로 정리해서 보여준다.

```
top_six_list <- moving_db %>%
    group_by(출발시군구코드) %>%
    summarize(이동인구_합_구 = sum(이동인구_합)) %>%
    slice_max(이동인구_합_구, n = 6) %>%
    select(출발시군구코드) %>%
    left_join(reference_db,
              by = c("출발시군구코드" = "시군구코드")) %>%
    select("시군구이름") %>%
    collect() %>% pull()
```

```
## Warning: Missing values are always removed in SQL.
## Use `SUM(x, na.rm = TRUE)` to silence this warning
```

지역별 5세 미만 아이들의 이동유형 차이

```
cummute_seoul <- moving_db %>%
    mutate(나이 = as.integer(나이)) %>%
    filter(나이 >= 70) %>%
    filter(이동유형 %in% c("HH", "HW", "WH", "WE")) %>%
    mutate(그룹이동시간 = 평균이동시간_분 * 이동인구_합) %>%
    group_by(출발시군구코드, 이동유형) %>%
    summarize(이동인구 = sum(이동인구_합, na.rm = TRUE),
              총이동시간 = sum(그룹이동시간, na.rm = TRUE)) %>%
    mutate(평균이동시간 = 총이동시간 / 이동인구) %>%
    left_join(reference_db,
              by = c("출발시군구코드" = "시군구코드")) %>%
    filter(시도코드 == 11000) %>%
    filter(시군구이름 %in% top_six_list) %>%
    ungroup() %>%
    select(시군구이름, 이동유형, 이동인구, 평균이동시간) %>%
    collect()
```

그래프 그려보기 (다음시간 미리보기)

```
cummute_seoul %<>%
    group_by(시군구이름) %>%
    mutate(이동유형 = factor(이동유형),
           시군구이름 = factor(시군구이름,
                           levels = top
    mutate(percentage = 이동인구 / sum(이동인
           .after = "이동인구")
```

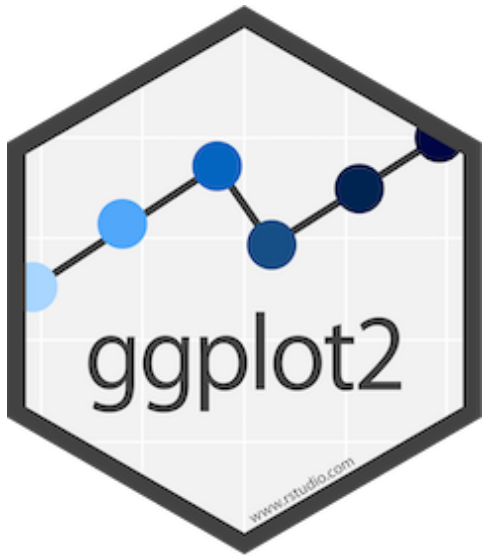
```
cummute_seoul %>%
  ggplot(aes(x = 이동유형,
             y = percentage * 100)) +
  geom_bar(stat = "identity",
    aes(fill = 이동유형)) +
  geom_label(aes(x = 이동유형,
    label = paste0(round(percentage * 100,
    facet_wrap(~시군구이름) +
    labs(title = "서울시 구별 아침 출근 시간 분
         subtitle = "이동 빈도 상위 6개구 (70.
    ylab("퍼센트 (%)") +
    xlab("이동 유형별 분류") +
    theme_bw()
```

시각화

레이어의 개념

대표적인 시각화 그래프들

꿀패키지 소개



같이 보면 좋은 책 추천

[1] R for Data Science

- 웹 상에 무료 공개된 책입니다.
- 위 교재의 한글 번역본 R 을 활용한 데이터과학도 있습니다.
- 도서 제목 클릭하셔서 구매하시면 저의 사리사욕을 충당하는데 도움이 됩니다.