빅데이터 소비 교재 : R 을 중심으로

- 필요패키지: `MASS`, `dplyr`, `ggvis`, `ca`
- 필요 데이터
    1. `card_trade2.csv`: 2011 년 11 월부터 2014 년 2 월까지의 신용카드 결제 기록 데이터
    2. `weather.csv`: 기상청에서 제공하는 지점별 기온데이터

학습목표

R 을 이용한 자료 타입에 따른 data manipulation 기법에 대해 학습하고, **dplyr** 패키지를 활용한 고급 데이터 핸들링 기법을 익힌다.

제공된 소비 데이터를 통해 소비 지출에 대한 패턴 분석을 한다.
탐색적 자료 분석 수준의 시각화 과정을 통해 월별, 분야별 평균 지출 비용과 서울 월별 평균 기온 사이의 연관성 분석을 포함한 분석 시나리오를 작성하고, 사용자의 월별 분야별 소비 패턴을 예측하는 모형을 구축해 활용성을 검증한다.

분할표(contingency table) 분석을 통해 카드사와 지출 분야 간 연관성에 대한 정보 및 시사점을 도출한다.
또한 대응일치분석(correspondence analysis)을 통해 카드사들의 포지셔닝에 대해 분석한다.

A. dplyr 패키지를 이용한 데이터 munging 맛보기

A.1 dplyr 패키지 소개

데이터 munging (또는 wrangling)은 전처리, 파싱, 필터링과 같이 데이터를 분석가가 원하는 형태로 변형하고 이리저리 핸들링하는 행위를 의미하는 신조어이다.

- Wiki 백과

dplyr 패키지는 하들리위컴(Hadley Wickham)이 최근 작성한 데이터 처리에 특화된 패키지이다.
하들리위컴이 같은 목적으로 작성한 패키지인 **plyr** 는 모든 함수가 R 로 작성되어 있어 속도가 느린 문제가 있었으나, **dplyr** 은 C++로 작성되었기 때문에 매우 빠른 속도로 데이터 처리를 수행할 수 있게 되었다.

dplyr 패키지를 이용하면 코드 작성이 직관적이고 가독성이 좋은 장점(특히 chaining syntax 를 이용 시)이 있다. 다만 **dplyr** 패키지 내에 몇몇 base 함수와 겹치는 이름의 함수가 있어 masking 이 일어남에 주의해야 한다.

- `filter()`, `lag()`, `intersect()`, `setdiff()`, `setequal()`, `union()` 등

plyr 패키지와 동시에 사용하는 경우 문제를 일으킬 수 있으므로 유의할 필요가 있다. 굳이 동시에 사용하려면 **plyr** 을 먼저 로드하기를 추천한다.

A.2 `card_trade.csv` 데이터 먼징

csv 파일로 주어진 데이터를 R 로 불러들일 때는 `read.csv()` 함수를 이용한다.
아래는 `card_trade2.csv` 라는 이름의 파일로 주어진 데이터를 R 로 불러들여 `card.trade` 라는 이름의 데이터프레임으로 저장하는 작업을 위한 코드이다. 불러온 데이터의 차원과 자료구조를 확인한다.

```
    card.trade <- read.csv("./Data/card_trade2.csv", fileEncoding="UTF-8")
    dim(card.trade)
```

```
## [1] 364068     42
```

```
    str(card.trade)
```

```
## 'data.frame':    364068 obs. of  42 variables:
##  $ U_TRADE_NO         : num  2.01e+19 2.01e+19 2.01e+19 2.01e+19 2.01e+19
 ...
##  $ USER_SID           : num  2.01e+19 2.01e+19 2.01e+19 2.01e+19 2.01e+19
 ...
##  $ CATEGORY_TYPE      : Factor w/ 14 levels "","CA","CB","CC",..: 3 2 6 2
10 7 2 2 14 14 ...
##  $ SMS_RECEIVE_DT     : num  2.01e+13 2.01e+13 2.01e+13 2.01e+13 2.01e+13
 ...
##  $ PAY_TYPE           : Factor w/ 4 levels "","PA01","PA03",..: 3 3 3 4 3
3 3 3 3 3 ...
##  $ TRADE_TYPE         : Factor w/ 4 levels "","TT01","TT02",..: 2 2 2 2 2
2 2 2 2 2 ...
##  $ PAY_CD             : Factor w/ 9 levels "","AM","BC","BK",..: 3 3 3 3 3
 3 3 3 3 3 ...
##  $ PAY_ACCOUNT        : Factor w/ 309 levels "","****","0",..: 83 83 83 83
 62 83 83 83 209 13 ...
##  $ TRADE_SITE_NM      : Factor w/ 66266 levels "","#1","#199-1 스테이크",..:
 14533 37365 24463 38677 34599 23190 17596 37365 10056 13285 ...
##  $ TRADE_MONEY        : int  48000 49000 19000 14000 253000 10000 80000 48
000 4600 5000 ...
##  $ TRADE_DT           : num  2.01e+13 2.01e+13 2.01e+13 2.01e+13 2.01e+13
 ...
##  $ QUOTA_MONTH        : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ BALANCE_MONEY      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ ADD_POINT          : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ USE_POINT          : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ SMS_ORG            : logi  NA NA NA NA NA NA ...
##  $ CAR_FILL_YN        : Factor w/ 3 levels "","N","Y": 2 2 2 2 2 2 2 2 2 2
 ...
##  $ ONLINE_SITE_YN     : Factor w/ 2 levels "","N": 2 2 2 2 2 2 2 2 2 2 ...
##  $ COMPANY_CARD_YN    : Factor w/ 2 levels "","N": 2 2 2 2 2 2 2 2 2 2 ...
##  $ INAREA_YN          : Factor w/ 3 levels "","N","Y": 2 2 2 2 2 2 2 2 2 2
 ...
##  $ FOREIGN_AMOUNT     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ FOREIGN_UNIT       : Factor w/ 21 levels "","AED","BRL",..: 14 14 14 14
 14 14 14 14 14 14 ...
##  $ MEMO               : Factor w/ 1095 levels "","  ARS","  INTERNET",..: 1
1 1 640 1 1 1 1 1 1 ...
##  $ PAY_NM             : Factor w/ 34 levels "","S C 체크",..: 28 28 28 29
28 28 28 28 28 28 ...
##  $ RESULT_PARSE       : int  0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ REG_DT             : Factor w/ 36 levels "","2013-12-11",..: 2 2 2 2 2
2 2 2 2 2 ...
##  $ TRADE_STAT         : Factor w/ 2 levels "","TS01": 2 2 2 2 2 2 2 2 2 2
 ...
##  $ USE_YN             : Factor w/ 2 levels "","Y": 2 2 2 2 2 2 2 2 2 2 ...
##  $ REG_TYPE           : Factor w/ 2 levels "","RG01": 2 2 2 2 2 2 2 2 2 2
 ...
##  $ SMS_SEND_ID        : logi  NA NA NA NA NA NA ...
##  $ REF_U_TRADE_NO     : logi  NA NA NA NA NA NA ...
##  $ MOD_DT             : logi  NA NA NA NA NA NA ...
##  $ USER_NM            : logi  NA NA NA NA NA NA ...
##  $ CHECK_MONEY        : int  0 0 0 0 0 0 0 0 0 100 ...
##  $ CARD_ADD_MONEY     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ NOTAPPLY_CHECK_YN  : Factor w/ 2 levels "","N": 2 2 2 2 2 2 2 2 2 2 ...
##  $ NOTAPPLY_CHECK_DT  : logi  NA NA NA NA NA NA ...
##  $ APP_YN             : Factor w/ 2 levels "","Y": 2 2 2 2 2 2 2 2 2 2 ...
##  $ AUTOPAY_TYPE       : Factor w/ 2 levels "","AP99": 2 2 2 2 2 2 2 2 2 2
 ...
##  $ AUTOPAY_REMAIND_CNT: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ JOIN_TYPE          : Factor w/ 2 levels "","JT02": 2 2 2 2 2 2 2 2 2 2
 ...
##  $ TRADE_YN           : Factor w/ 3 levels "","N","Y": 3 3 3 3 3 3 3 3 3 3
 ...
```

분석에 사용될 주요 변수는 다음과 같다.

- CATEGORY_TYPE: 신용카드 사용액 지출 분야
  - CA: 마트/쇼핑, CB: 외식/부식, CC: 커피/간식, CD: 레저/문화, CE: 건강/의료, CF: 미용/뷰티, CG: 교통/주유, CH: 주거/생활, CI: 교육/학원, CJ: 보험/세금, CK: 외환/해외, CL:기타, CZ: 미지정
- PAY_CD: 신용카드사 이름
- FOREIGN_UNIT: 결제환 종류

`library()` 함수를 이용해 **dplyr** 패키지를 로딩해보자.

```
    library(dplyr)
```

```
##
## Attaching package: 'dplyr'
##
## The following objects are masked from 'package:stats':
##
##     filter, lag
##
## The following objects are masked from 'package:base':
```

```
##
##     intersect, setdiff, setequal, union
```

관측치의 개수가 많은 경우 데이터프레임을 다룰 때에는 **dplyr** 패키지에서 제공하는 `tbl_df()` 함수를 이용해 로컬데이터프레임으로 변환해 사용하는 것이 좋다.
로컬 데이터프레임의 개념은 대규모의 데이터프레임에 대해 데이터 내용을 화면 출력이 잘 되도록 임시로 랩을 씌운 것이라 이해하면 된다.
`tbl_df()` 함수는 큰 규모의 데이터프레임을 보기 쉬운 형태로 바꿔주는데 특히 모든 데이터를 화면에 출력하는 실수 방지에 유용하다. 필요한 경우 `as.data.frame()` 함수를 이용해 원래의 데이터프레임으로 되돌릴 수 있다.

- (보통은 실수로) 큰 규모의 데이터프레임 이름을 콘솔에 입력하는 경우 데이터 내용물이 화면을 가득 메우는 동안 하염없이 기다리거나 컴퓨터가 힘들어 하는 것을 경험하게 됨

```
    card.trade    ### Not run...
```

- 큰 규모의 데이터프레임이라 할지라도 `tbl_df()`로 변환한 데이터프레임 이름을 콘솔에 입력하면 별 어려움 없이 내용물을 살짝 들여다볼 수 있음

```
    card.df <- tbl_df(card.trade)
    card.df
```

```
## Source: local data frame [364,068 x 42]
##
##    U_TRADE_NO USER_SID CATEGORY_TYPE SMS_RECEIVE_DT PAY_TYPE TRADE_TYPE
##         (dbl)    (dbl)        (fctr)          (dbl)   (fctr)     (fctr)
## 1    2.01e+19 2.01e+19            CB       2.01e+13     PA03       TT01
## 2    2.01e+19 2.01e+19            CA       2.01e+13     PA03       TT01
## 3    2.01e+19 2.01e+19            CE       2.01e+13     PA03       TT01
## 4    2.01e+19 2.01e+19            CA       2.01e+13     PA12       TT01
## 5    2.01e+19 2.01e+19            CI       2.01e+13     PA03       TT01
## 6    2.01e+19 2.01e+19            CF       2.01e+13     PA03       TT01
## 7    2.01e+19 2.01e+19            CA       2.01e+13     PA03       TT01
## 8    2.01e+19 2.01e+19            CA       2.01e+13     PA03       TT01
## 9    2.01e+19 2.01e+19            CZ       2.01e+13     PA03       TT01
## 10   2.01e+19 2.01e+19            CZ       2.01e+13     PA03       TT01
## ..         ...      ...           ...            ...      ...        ...
## Variables not shown: PAY_CD (fctr), PAY_ACCOUNT (fctr), TRADE_SITE_NM
##   (fctr), TRADE_MONEY (int), TRADE_DT (dbl), QUOTA_MONTH (int),
##   BALANCE_MONEY (int), ADD_POINT (int), USE_POINT (int), SMS_ORG (lgl),
##   CAR_FILL_YN (fctr), ONLINE_SITE_YN (fctr), COMPANY_CARD_YN (fctr),
##   INAREA_YN (fctr), FOREIGN_AMOUNT (dbl), FOREIGN_UNIT (fctr), MEMO
##   (fctr), PAY_NM (fctr), RESULT_PARSE (int), REG_DT (fctr), TRADE_STAT
##   (fctr), USE_YN (fctr), REG_TYPE (fctr), SMS_SEND_ID (lgl),
```

```
##   REF_U_TRADE_NO (lgl), MOD_DT (lgl), USER_NM (lgl), CHECK_MONEY (int),
##   CARD_ADD_MONEY (int), NOTAPPLY_CHECK_YN (fctr), NOTAPPLY_CHECK_DT (lgl),
##   APP_YN (fctr), AUTOPAY_TYPE (fctr), AUTOPAY_REMAIND_CNT (int), JOIN_TYPE
##   (fctr), TRADE_YN (fctr)
```

- 불필요한 메모리 사용을 막기 위해 card.trade 를 현재 사용환경에서 제거

```
rm(card.trade)
```

dplyr 패키지의 주요 함수는 다음과 같다.

- filter(): 조건에 맞는 행 추출 (엑셀의 필터 기능과 유사)
- select(): 원하는 변수(열) 추출
- arrange(): 정렬하기(sorting)
- mutate(): 기존 변수를 이용해 새로운 변수 생성
- summarise(): 기존 변수를 가지고 계산해 하나의 값으로 내놓기. group_by()와 함께 사용하면 강력

filter() 함수는 지정한 조건에 따라 행(row)들을 추출하는데 사용한다.
데이터프레임 이름을 반복적으로 사용하지 않고 조건에 맞는 행을 추출할 수 있어 코드가 간결해지기 때문에 코드 작성의 편의성 및 가독성을 향상시킬 수 있다.

- 행 이름을 사용한 데이터프레임에 적용하는 경우 행이름은 유지되지 않음에 유의
- 여러 조건에 대한 AND 연산은 콤마(,)로, OR 연산은 세로막대(|)로 표현
- 아래 코드는 card.df 에서 마트/쇼핑 분야(CA)에 원화(KRW)로 결제한 데이터를 추출하되, 데이터 내에 포함된 시험용 결제 데이터는 제외하는 예임

```
CA.KRW <- filter(card.df,
                 CATEGORY_TYPE=="CA",           # , AND
                 FOREIGN_UNIT=="KRW",           # , AND
                 TRADE_SITE_NM!="테스트" |      # | OR
                   TRADE_SITE_NM!="시험 1" |    # | OR
                   TRADE_SITE_NM!="시험 2" |    # | OR
```

```
                        TRADE_SITE_NM!="시험 3" |      # | OR
                        TRADE_SITE_NM!="시험 4" )
```

`select()` 함수는 지정한 조건에 따라 열(column)들을 추출하는데 사용한다.
Base R 에서 인덱스벡터를 사용할 때 코드가 지저분해지는 경향이 있는데 `select()`을 이용하면 코드가 깔끔해지는 효과가 있다.
물론 대용량 데이터를 다루는 경우 처리 속도도 향상된다.

- `filter()` 함수와 같은 형식의 문법 사용
- SQL 의 SELECT 와 같은 기능
- 아래는 위에서 생성한 새로운 데이터프레임에서 일부 변수(CATEGORY_TYPE, TRADE_TYPE, PAY_CD, FOREIGN_UNIT, TRADE_MONEY)만 추출한 데이터프레임 생성해 dat 라는 이름의 데이터프레임으로 저장하는 코드임

```
    dat <- select(CA.KRW,
                  CATEGORY_TYPE,
                  TRADE_TYPE,
                  PAY_CD,
                  FOREIGN_UNIT,
                  TRADE_MONEY)
```

- 이 데이터프레임에서 - 연산자를 이용해 FOREIGN_UNIT 변수 제거

```
    select(dat, -FOREIGN_UNIT)
```

```
## Source: local data frame [125,955 x 4]
##
##    CATEGORY_TYPE TRADE_TYPE PAY_CD TRADE_MONEY
##           (fctr)     (fctr) (fctr)       (int)
## 1             CA       TT01     BC       49000
## 2             CA       TT01     BC       14000
## 3             CA       TT01     BC       80000
## 4             CA       TT01     BC       48000
## 5             CA       TT01     SH        1750
## 6             CA       TT01     SH        3100
## 7             CA       TT01     SH         543
## 8             CA       TT01     SH        4480
## 9             CA       TT01     SH        8020
## 10            CA       TT01     SH       35000
## ..           ...        ...    ...         ...
```

- 변수 이름을 수정하는 것도 가능함

```
    select(dat, CURRENCY = FOREIGN_UNIT)
```

```
## Source: local data frame [125,955 x 1]
##
##    CURRENCY
##      (fctr)
## 1       KRW
## 2       KRW
## 3       KRW
## 4       KRW
## 5       KRW
## 6       KRW
## 7       KRW
## 8       KRW
## 9       KRW
## 10      KRW
## ..      ...
```

- Helper functions: `contains()`, `matches()`, `starts_with()`, `ends_with()`

```
    select(dat, contains("PAY"))    # 변수명에 'PAY'를 포함하는 변수를 모두 추출
```

```
## Source: local data frame [125,955 x 1]
##
##    PAY_CD
##    (fctr)
## 1      BC
## 2      BC
## 3      BC
## 4      BC
## 5      SH
## 6      SH
## 7      SH
## 8      SH
## 9      SH
## 10     SH
## ..    ...
```

```
    select(dat, matches(".t."))     # regular expression 에 부합하는 이름의 변수 추출
```

```
## Source: local data frame [125,955 x 2]
##
##    CATEGORY_TYPE TRADE_TYPE
##           (fctr)     (fctr)
## 1             CA       TT01
## 2             CA       TT01
## 3             CA       TT01
## 4             CA       TT01
```

```
## 5              CA       TT01
## 6              CA       TT01
## 7              CA       TT01
## 8              CA       TT01
## 9              CA       TT01
## 10             CA       TT01
## ..            ...        ...
```

```
    select(dat, starts_with("PAY")) # 'PAY'로 시작하는 이름의 변수 추출
```

```
## Source: local data frame [125,955 x 1]
##
##    PAY_CD
##    (fctr)
## 1      BC
## 2      BC
## 3      BC
## 4      BC
## 5      SH
## 6      SH
## 7      SH
## 8      SH
## 9      SH
## 10     SH
## ..    ...
```

```
    select(dat, ends_with("TYPE"))  # 'TYPE'으로 끝나는 이름의 변수 추출
```

```
## Source: local data frame [125,955 x 2]
##
##    CATEGORY_TYPE TRADE_TYPE
##           (fctr)     (fctr)
## 1             CA       TT01
## 2             CA       TT01
## 3             CA       TT01
## 4             CA       TT01
## 5             CA       TT01
## 6             CA       TT01
## 7             CA       TT01
## 8             CA       TT01
## 9             CA       TT01
## 10            CA       TT01
## ..           ...        ...
```

arrage() 함수를 이용해 자료를 정렬해보자. 아래는 dat 을 TRADE_MONEY 의 크기 순으로 정렬하는 코드이다.

```
    arrange(dat, TRADE_MONEY)
```

```
## Source: local data frame [125,955 x 5]
##
##    CATEGORY_TYPE TRADE_TYPE PAY_CD FOREIGN_UNIT TRADE_MONEY
##           (fctr)     (fctr) (fctr)       (fctr)       (int)
## 1             CA       TT02     SH          KRW           1
## 2             CA       TT02     BC          KRW           1
## 3             CA       TT01     SH          KRW           2
## 4             CA       TT01     SH          KRW           2
## 5             CA       TT01     SH          KRW           2
## 6             CA       TT01     SH          KRW           5
## 7             CA       TT01     SH          KRW           7
## 8             CA       TT02     BC          KRW          10
## 9             CA       TT01     SH          KRW          13
## 10            CA       TT01     SH          KRW          13
## ..           ...        ...    ...          ...         ...
```

```
    arrange(dat, desc(TRADE_MONEY)) # 내림차순 정렬
```

```
## Source: local data frame [125,955 x 5]
##
##    CATEGORY_TYPE TRADE_TYPE PAY_CD FOREIGN_UNIT TRADE_MONEY
##           (fctr)     (fctr) (fctr)       (fctr)       (int)
## 1             CA       TT01     BK          KRW    34402090
## 2             CA       TT01     BK          KRW     9000000
## 3             CA       TT01     KM          KRW     6710000
## 4             CA       TT02     KM          KRW     6710000
## 5             CA       TT01     KM          KRW     5000000
## 6             CA       TT01     BC          KRW     4770000
## 7             CA       TT02     BC          KRW     4770000
## 8             CA       TT01     BC          KRW     4500000
## 9             CA       TT01     KM          KRW     3344000
## 10            CA       TT02     KM          KRW     3344000
## ..           ...        ...    ...          ...         ...
```

mutate() 함수를 이용해 열을 변형 또는 추가하는 것이 가능하다. 아래는 변수 TRADE_MONEY 를 1,000 으로 나눈 값으로 변형하는 예이다.

```
    mutate(card.df, TRADE_MONEY=TRADE_MONEY/1000)
```

```
## Source: local data frame [364,068 x 42]
##
##    U_TRADE_NO USER_SID CATEGORY_TYPE SMS_RECEIVE_DT PAY_TYPE TRADE_TYPE
##         (dbl)    (dbl)        (fctr)          (dbl)   (fctr)     (fctr)
## 1    2.01e+19 2.01e+19            CB       2.01e+13     PA03       TT01
## 2    2.01e+19 2.01e+19            CA       2.01e+13     PA03       TT01
## 3    2.01e+19 2.01e+19            CE       2.01e+13     PA03       TT01
## 4    2.01e+19 2.01e+19            CA       2.01e+13     PA12       TT01
## 5    2.01e+19 2.01e+19            CI       2.01e+13     PA03       TT01
## 6    2.01e+19 2.01e+19            CF       2.01e+13     PA03       TT01
```

```
## 7    2.01e+19 2.01e+19            CA       2.01e+13     PA03       TT01
## 8    2.01e+19 2.01e+19            CA       2.01e+13     PA03       TT01
## 9    2.01e+19 2.01e+19            CZ       2.01e+13     PA03       TT01
## 10   2.01e+19 2.01e+19            CZ       2.01e+13     PA03       TT01
## ..        ...      ...           ...            ...      ...        ...
## Variables not shown: PAY_CD (fctr), PAY_ACCOUNT (fctr), TRADE_SITE_NM
##   (fctr), TRADE_MONEY (dbl), TRADE_DT (dbl), QUOTA_MONTH (int),
##   BALANCE_MONEY (int), ADD_POINT (int), USE_POINT (int), SMS_ORG (lgl),
##   CAR_FILL_YN (fctr), ONLINE_SITE_YN (fctr), COMPANY_CARD_YN (fctr),
##   INAREA_YN (fctr), FOREIGN_AMOUNT (dbl), FOREIGN_UNIT (fctr), MEMO
##   (fctr), PAY_NM (fctr), RESULT_PARSE (int), REG_DT (fctr), TRADE_STAT
##   (fctr), USE_YN (fctr), REG_TYPE (fctr), SMS_SEND_ID (lgl),
##   REF_U_TRADE_NO (lgl), MOD_DT (lgl), USER_NM (lgl), CHECK_MONEY (int),
##   CARD_ADD_MONEY (int), NOTAPPLY_CHECK_YN (fctr), NOTAPPLY_CHECK_DT (lgl),
##   APP_YN (fctr), AUTOPAY_TYPE (fctr), AUTOPAY_REMAIND_CNT (int), JOIN_TYPE
##   (fctr), TRADE_YN (fctr)
```

summarise() 함수는 데이터 내용을 지정한 방법대로 집계해 하나의 벡터를 생성해 주는 함수이다. 아래는 card.df 의 TRADE_MONEY 의 평균값과 건수를 계산하는 예이다.

```
    summarise(card.df, Avg=mean(TRADE_MONEY, na.rm=T), Transactions=n())
```

```
## Source: local data frame [1 x 2]
##
##        Avg Transactions
##      (dbl)        (int)
## 1 71167.62       364068
```

- n()는 건수를 계수

group_by() 함수를 이용하면 그룹별 분석이 가능하다. 아래는 card.df 를 CATEGORY_TYPE 별로 구분해 TRADE_MONEY 의 평균, 표준편차, 건수를 계산하는 예이다.

```
    card.df.grp <- group_by(card.df, CATEGORY_TYPE)
    summarise(card.df.grp,
              Average=mean(TRADE_MONEY),
              Stdev=sd(TRADE_MONEY),
              Transactions=n()
              )
```

```
## Source: local data frame [14 x 4]
##
##    CATEGORY_TYPE   Average      Stdev Transactions
##           (fctr)     (dbl)      (dbl)        (int)
## 1                      NA        NaN         3700
## 2            CA  39502.08  148431.48       125978
## 3            CB  28509.79   45978.18        40660
## 4            CC  11141.17   55631.89        18710
## 5            CD  60809.74  287714.77        12275
```

```
## 6             CE  44758.25  318929.15        15810
## 7             CF  39782.64  115082.60         3575
## 8             CG  47719.63  463324.23        36489
## 9             CH  59686.37   95303.03         5596
## 10            CI 179271.74  173358.70         1656
## 11            CJ 282149.42 1724431.86         5556
## 12            CK  93585.08  318525.48         3604
## 13            CL  10984.68   15520.93          912
## 14            CZ 149855.91 1488560.94        89547
```

- `n()`는 그룹 내 해당 건수를 계수
- (c.f.) `n_distinct()`는 겹치지 않는 건수를 계수

summarise_each() 함수를 이용하는 방법도 있다. 아래는 card.df 의 TRADE_MONEY 의 평균, 표준편차, 건수를 CATEGORY_TYPE 별로 계산하는 예이다.

```
    summarise_each(card.df.grp,
                   funs(mean(., na.rm=T), sd(., na.rm=T), n()),
                   matches("TRADE_MONEY")
                   )
```

```
## Source: local data frame [14 x 4]
##
##    CATEGORY_TYPE      mean         sd      n
##           (fctr)     (dbl)      (dbl)  (int)
## 1                       NA         NA   3700
## 2             CA  39502.08  148431.48 125978
## 3             CB  28509.79   45978.18  40660
## 4             CC  11141.17   55631.89  18710
## 5             CD  60809.74  287714.77  12275
## 6             CE  44758.25  318929.15  15810
## 7             CF  39782.64  115082.60   3575
## 8             CG  47719.63  463324.23  36489
## 9             CH  59686.37   95303.03   5596
## 10            CI 179271.74  173358.70   1656
## 11            CJ 282149.42 1724431.86   5556
## 12            CK  93585.08  318525.48   3604
## 13            CL  10984.68   15520.93    912
## 14            CZ 149855.91 1488560.94  89547
```

Chaining 또는 Pipelining 은 아마도 **dplyr** 패키지에서 가장 유용한 도구일 것이다.
%>%를 이용하면 최종 분석 결과를 얻기 위해 임시 데이터프레임을 만들 필요가 없이 연결된 여러 작업을 %>% (infix 연산자, 'then'으로 읽음)을 사용해 순서대로 나열할 수 있다.
코드 작성의 편의성 및 가독성을 어마어마하게 향상시킬 수 있다. dplyr 패키지는 magrittr 패키지로부터 %>% 연산자를 불러와서 사용하기 때문에 다른 종류의 nesting 에도 활용 가능하다.

```
    x <- 1:3; y <- 2:4
```

- $\sqrt{\sum_{i=1}^{3}(x_i - y_i)^2}$ 계산하기

```
    sqrt(sum((x-y)^2))       # 눈이 어질어질...
```

```
## [1] 1.732051
```

```
    (x-y)^2 %>% sum() %>% sqrt()
```

```
## [1] 1.732051
```

- 위의 예에서는 card.df.grp 라는 이름의 데이터프레임을 추가로 생성했지만 %>%을 사용하면 아래와 같이 간명해짐

```
    card.df %>%
      group_by(CATEGORY_TYPE) %>%
      summarise(
        Average=mean(TRADE_MONEY),
        Stdev=sd(TRADE_MONEY),
        Transactions=n()
      )
```

```
## Source: local data frame [14 x 4]
##
##    CATEGORY_TYPE   Average      Stdev Transactions
##           (fctr)     (dbl)      (dbl)        (int)
## 1                      NA        NaN         3700
## 2             CA  39502.08  148431.48       125978
## 3             CB  28509.79   45978.18        40660
## 4             CC  11141.17   55631.89        18710
## 5             CD  60809.74  287714.77        12275
## 6             CE  44758.25  318929.15        15810
## 7             CF  39782.64  115082.60         3575
## 8             CG  47719.63  463324.23        36489
## 9             CH  59686.37   95303.03         5596
## 10            CI 179271.74  173358.70         1656
## 11            CJ 282149.42 1724431.86         5556
## 12            CK  93585.08  318525.48         3604
## 13            CL  10984.68   15520.93          912
## 14            CZ 149855.91 1488560.94        89547
```

sample_n(), sample_frac() 함수를 이용해 랜덤하게 일부 행을 추출할 수 있다. 추출할 행의 개수를 지정하는 것도 가능하고 추출 비율을 지정할 수도 있다.

```
    sample_n(card.df, 10, replace=F) # 랜덤하게 10 개 행 추출
```

```
## Source: local data frame [10 x 42]
##
##    U_TRADE_NO USER_SID CATEGORY_TYPE SMS_RECEIVE_DT PAY_TYPE TRADE_TYPE
```

```
##         (dbl)    (dbl)        (fctr)          (dbl)   (fctr)     (fctr)
## 1    2.01e+19 2.01e+19            CC       2.01e+13     PA12       TT01
## 2    2.01e+19 2.01e+19            CA       2.01e+13     PA12       TT01
## 3    2.01e+19 2.01e+19            CE       2.01e+13     PA12       TT01
## 4    2.01e+19 2.01e+19            CC       2.01e+13     PA12       TT01
## 5    2.01e+19 2.01e+19            CG       2.01e+13     PA12       TT01
## 6    2.01e+19 2.01e+19            CE       2.01e+13     PA12       TT01
## 7    2.01e+19 2.01e+19            CZ       2.01e+13     PA12       TT01
## 8    2.01e+19 2.01e+19            CZ       2.01e+13     PA12       TT01
## 9    2.01e+19 2.01e+19            CZ       2.01e+13     PA12       TT01
## 10   2.01e+19 2.01e+19            CC       2.01e+13     PA12       TT01
## Variables not shown: PAY_CD (fctr), PAY_ACCOUNT (fctr), TRADE_SITE_NM
##   (fctr), TRADE_MONEY (int), TRADE_DT (dbl), QUOTA_MONTH (int),
##   BALANCE_MONEY (int), ADD_POINT (int), USE_POINT (int), SMS_ORG (lgl),
##   CAR_FILL_YN (fctr), ONLINE_SITE_YN (fctr), COMPANY_CARD_YN (fctr),
##   INAREA_YN (fctr), FOREIGN_AMOUNT (dbl), FOREIGN_UNIT (fctr), MEMO
##   (fctr), PAY_NM (fctr), RESULT_PARSE (int), REG_DT (fctr), TRADE_STAT
##   (fctr), USE_YN (fctr), REG_TYPE (fctr), SMS_SEND_ID (lgl),
##   REF_U_TRADE_NO (lgl), MOD_DT (lgl), USER_NM (lgl), CHECK_MONEY (int),
##   CARD_ADD_MONEY (int), NOTAPPLY_CHECK_YN (fctr), NOTAPPLY_CHECK_DT (lgl),
##   APP_YN (fctr), AUTOPAY_TYPE (fctr), AUTOPAY_REMAIND_CNT (int), JOIN_TYPE
##   (fctr), TRADE_YN (fctr)

    sample_frac(card.df, 0.01)       # 랜덤하게 1 퍼센트(fraction=0.01)의 행을
선택해 추출

## Source: local data frame [3,641 x 42]
##
##    U_TRADE_NO USER_SID CATEGORY_TYPE SMS_RECEIVE_DT PAY_TYPE TRADE_TYPE
##         (dbl)    (dbl)        (fctr)          (dbl)   (fctr)     (fctr)
## 1    2.01e+19 2.01e+19            CZ       2.01e+13     PA12       TT01
## 2    2.01e+19 2.01e+19            CA       2.01e+13     PA12       TT01
## 3    2.01e+19 2.01e+19            CB       2.01e+13     PA03       TT01
## 4    2.01e+19 2.01e+19            CZ       2.01e+13     PA12       TT01
## 5    2.01e+19 2.01e+19            CC       2.01e+13     PA12       TT01
## 6    2.01e+19 2.01e+19            CD       2.01e+13     PA12       TT01
## 7    2.01e+19 2.01e+19            CZ       2.01e+13     PA12       TT03
## 8    2.01e+19 2.01e+19            CZ       2.01e+13     PA12       TT01
## 9    2.01e+19 2.01e+19            CB       2.01e+13     PA03       TT01
## 10   2.01e+19 2.01e+19            CA       2.01e+13     PA12       TT02
## ..        ...      ...           ...           ...      ...        ...
## Variables not shown: PAY_CD (fctr), PAY_ACCOUNT (fctr), TRADE_SITE_NM
##   (fctr), TRADE_MONEY (int), TRADE_DT (dbl), QUOTA_MONTH (int),
##   BALANCE_MONEY (int), ADD_POINT (int), USE_POINT (int), SMS_ORG (lgl),
##   CAR_FILL_YN (fctr), ONLINE_SITE_YN (fctr), COMPANY_CARD_YN (fctr),
##   INAREA_YN (fctr), FOREIGN_AMOUNT (dbl), FOREIGN_UNIT (fctr), MEMO
##   (fctr), PAY_NM (fctr), RESULT_PARSE (int), REG_DT (fctr), TRADE_STAT
##   (fctr), USE_YN (fctr), REG_TYPE (fctr), SMS_SEND_ID (lgl),
```

```
##   REF_U_TRADE_NO (lgl), MOD_DT (lgl), USER_NM (lgl), CHECK_MONEY (int),
##   CARD_ADD_MONEY (int), NOTAPPLY_CHECK_YN (fctr), NOTAPPLY_CHECK_DT (lgl),
##   APP_YN (fctr), AUTOPAY_TYPE (fctr), AUTOPAY_REMAIND_CNT (int), JOIN_TYPE
##   (fctr), TRADE_YN (fctr)
```

A.3 기상 데이터 munging

weather.csv 데이터를 로드해 weather.df 라는 이름의 데이터프레임으로 저장한 후 데이터 구조를 확인해보자.

```
    weather <- read.csv("./Data/weather.csv", fileEncoding="UTF-8")
    weather.df <- tbl_df(weather)
    str(weather.df)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    691920 obs. of  4 variables:
##  $ 지역    : Factor w/ 93 levels "강릉(청)","강진군(공)",..: 27 27 27 27 27
27 27 27 27 27 ...
##  $ 기상구분: Factor w/ 2 levels "강수량","평균기온": 1 1 1 1 1 1 1 1 1 1 ...
##  $ 측정값  : num  NA 3 NA 8.5 37.5 NA NA 3 0.4 NA ...
##  $ 일자    : int  20100602 20100702 20100802 20100902 20101002 20101102
20101202 20100103 20100203 20100303 ...
```

```
    rm(weather)
```

filter() 함수를 이용해 서울 지역의 평균기온 정보만 추출해 Seoul.weather 라는 이름으로 저장하자. 결측치를 포함하는 행은 na.omit()를 이용해 제거한다.

```
    Seoul.weather <- filter(weather.df, 지역 == '서울(청)', 기상구분 == '평균기
온') %>%
      na.omit()
      # na.omit(filter(weather.df, 지역 == '서울(청)', 기상구분 == '평균기온'))
 와 동일

    Seoul.weather
```

```
## Source: local data frame [3,444 x 4]
##
##        지역 기상구분 측정값     일자
##      (fctr)   (fctr)  (dbl)    (int)
## 1  서울(청) 평균기온   -6.8 20110101
## 2  서울(청) 평균기온   -0.2 20110201
```

```
## 3  서울(청) 평균기온    0.5 20110301
## 4  서울(청) 평균기온    9.1 20110401
## 5  서울(청) 평균기온   12.5 20110501
## 6  서울(청) 평균기온   18.0 20110601
## 7  서울(청) 평균기온   25.1 20110701
## 8  서울(청) 평균기온   25.6 20110801
## 9  서울(청) 평균기온   27.0 20110901
## 10 서울(청) 평균기온   12.7 20111001
## ..      ...      ...    ...      ...
```

데이터 내에 포함된 날짜 정보 Seoul.weather$일자가 정수형 변수로 주어졌기 때문에 이를 Date 타입의 변수로 변환해 보자. 아래는 해당 변수를 문자형 변수로 바꾼 후 as.Date()를 이용해 날짜형 변수로 변환하는 코드이다.

```
    class(Seoul.weather$일자) <- "character"
    Seoul.weather$일자 <- as.Date(Seoul.weather$일자, format="%Y%m%d")

    unique(format(Seoul.weather$일자, "%Y%m")) # 데이터가 커버하는 일자 범위 확
인
```

```
##  [1] "201101" "201102" "201103" "201104" "201105" "201106" "201107"
##  [8] "201108" "201109" "201110" "201111" "201112" "201201" "201202"
## [15] "201203" "201204" "201205" "201206" "201207" "201208" "201209"
## [22] "201210" "201211" "201212" "201301" "201302" "201303" "201304"
## [29] "201305" "201306" "201307" "201308" "201309" "201310" "201311"
## [36] "201312" "201401" "201402" "201403" "201404" "201405" "201406"
## [43] "201407" "201408" "201409" "201001" "201002" "201003" "201004"
## [50] "201005" "201006" "201007" "201008" "201009" "201010" "201011"
## [57] "201012"
```

- unique() 함수는 중복된 값은 제거하고 내용물을 보여줌
- format() 함수는 Date 타입의 변수를 지정한 포맷을 적용함.

마찬가지 방법으로 card.df 의 날짜 정보를 정리해보자.

```
    card.df$TRADE_DT <- as.Date(as.character(card.df$TRADE_DT/1000000 + 1), f
ormat="%Y%m%d")
    sort(unique(format(card.df$TRADE_DT, "%Y%m")))
```

```
##  [1] "201110" "201111" "201112" "201201" "201202" "201203" "201204"
##  [8] "201205" "201206" "201207" "201208" "201209" "201210" "201211"
## [15] "201212" "201301" "201302" "201303" "201304" "201305" "201306"
```

```
## [22] "201307" "201308" "201309" "201310" "201311" "201312" "201401"
## [29] "201402"
```

2013 년 정보만을 추출해보자.

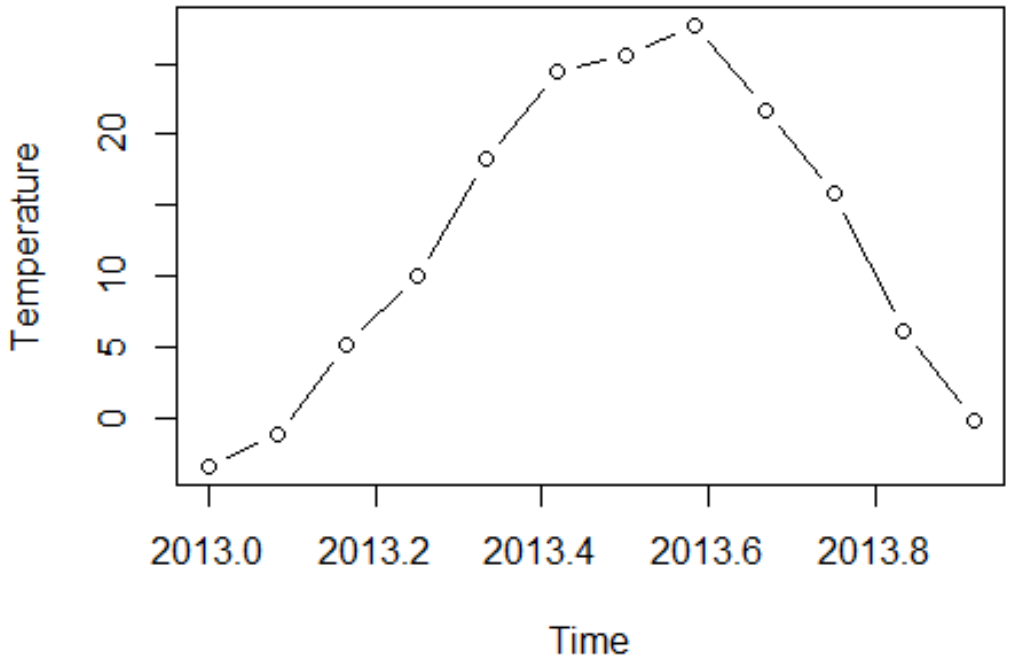
```
    card.2013 <- card.df %>%
      filter(format(TRADE_DT, "%Y%m") >= '201301',
             format(TRADE_DT, "%Y%m") <= '201312',
             FOREIGN_UNIT == "KRW",
             TRADE_SITE_NM != "테스트" |
               TRADE_SITE_NM != "시험 1" |
               TRADE_SITE_NM != "시험 2" |
               TRADE_SITE_NM != "시험 3" |
               TRADE_SITE_NM != "시험 4"
             ) %>%
      group_by(format(TRADE_DT, "%Y%m")) %>%
      summarise(Average=mean(as.double(TRADE_MONEY), na.rm=T), N=n())

    card.2013
```

```
## Source: local data frame [12 x 3]
##
##    format(TRADE_DT, "%Y%m")   Average     N
##                       (chr)     (dbl) (int)
## 1                    201301  85798.25  1705
## 2                    201302 115817.45  1671
## 3                    201303 124336.54  4195
## 4                    201304  78284.69  5847
## 5                    201305 124459.88  8067
## 6                    201306  87581.80 11519
## 7                    201307  72706.63 14859
## 8                    201308 104022.57 19513
## 9                    201309  58649.39 23899
## 10                   201310  61874.97 30726
## 11                   201311  65533.07 37718
## 12                   201312  60564.83 51871
```

위에서 생성한 card.2013 자료와의 결합을 고려해 2013 년의 서울 지역 월별 평균 기온을 산출해보자.

```
    Seoul.temp <- Seoul.weather %>%
      filter(format(일자, "%Y%m") >= '201301',
             format(일자, "%Y%m") <= '201312') %>%
      group_by(format(일자, "%Y%m")) %>%
      summarise(Average=mean(측정값, na.rm=T))
```

```r
Seoul.temp <- ts(Seoul.temp$Average, start=c(2013,1), frequency=12)
plot(Seoul.temp, ylab="Temperature", type='b')
```



- `ts()` 함수는 데이터를 시계열 타입으로 바꿔줌
- `plot.ts()` 함수는 시계열 도표 작성을 해줌

B. ggvis 패키지를 이용한 데이터 시각화

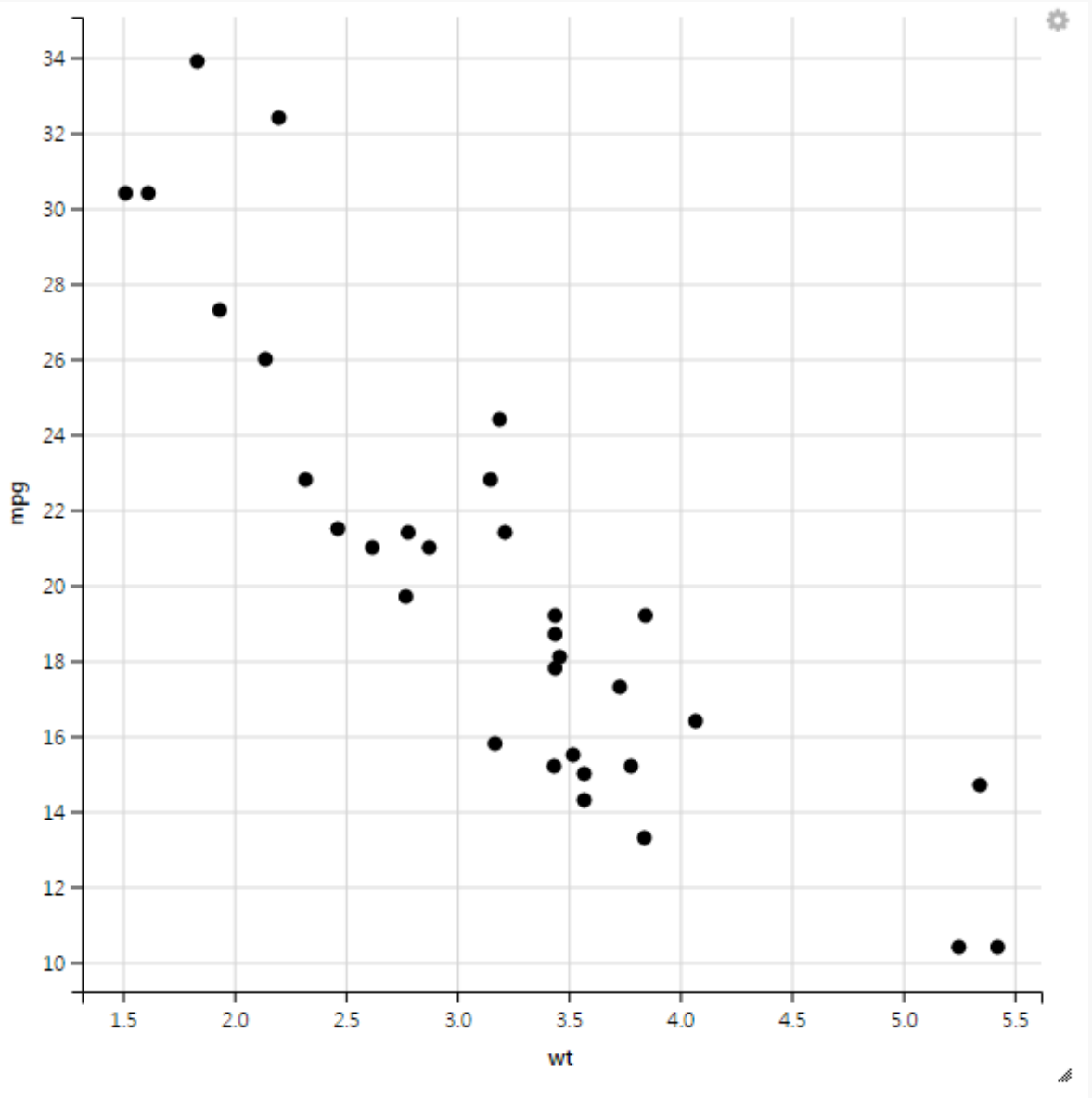
ggvis 패키지는 탐색적 자료분석을 위한 인터랙티브 그래픽 작업에 유용한 패키지이다.
ggplot2 패키지와 비슷하지만 보다 인터랙티브한 특성을 갖고 있다.
또한 shiny 패키지의 리액티브한 프로그래밍 모델, dplyr 패키지의 데이터 변형 문법과 잘 어울린다.

ggvis 패키지로 작성한 그래픽은 기본적으로 웹 그래픽이기 때문에 정적(static)인 특성을 가지고 있는 전통적 R 그래픽과 매우 다르다.
특히, 인터랙티브한 성격이 오히려 보고서 작성이나 출판에는 다소 불편을 초래하는 면이 있어, 이 부분에 대한 개선 작업이 계속 되고 있다고 한다.

B.1 ggvis() 함수

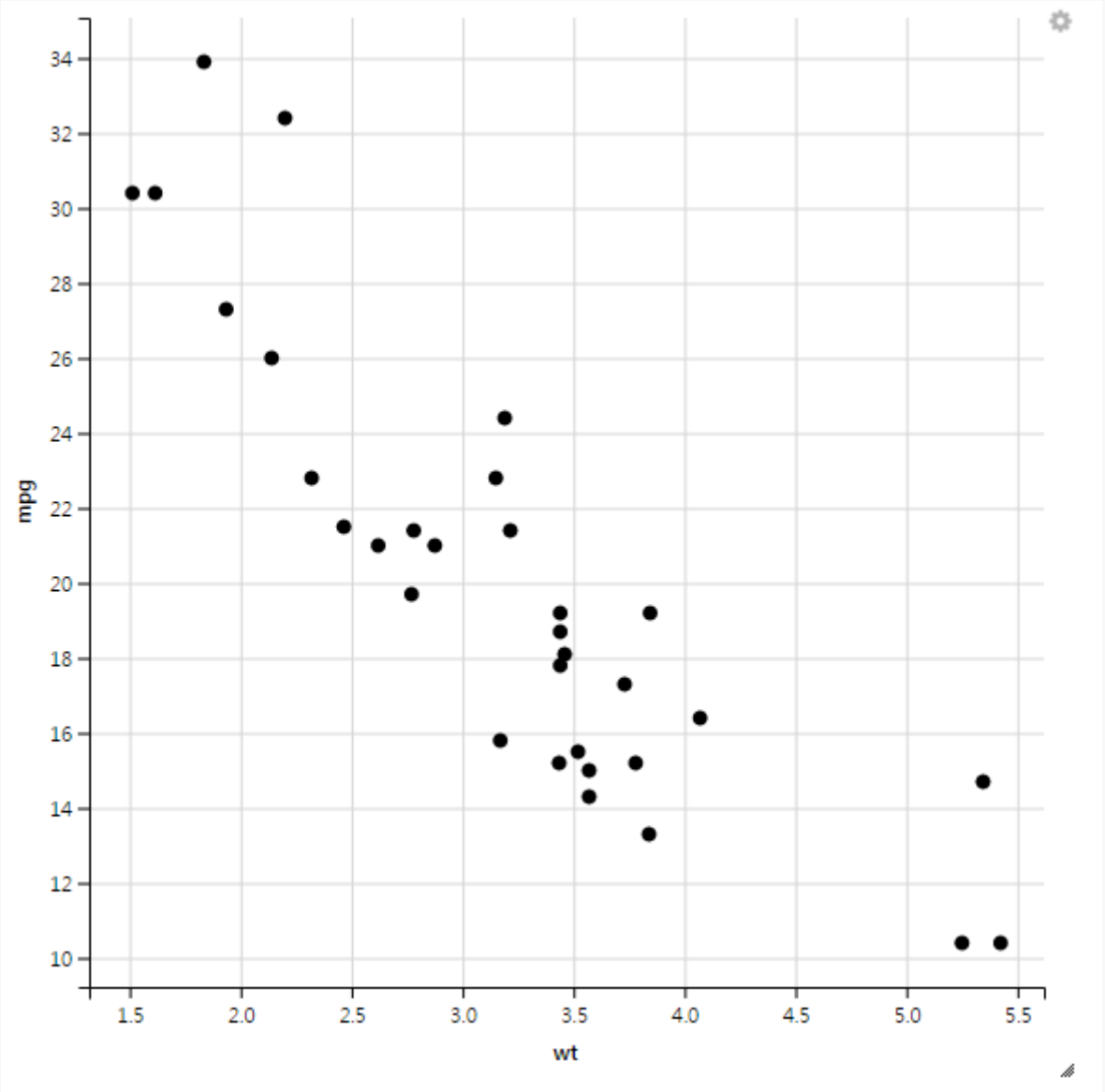
- ggvis() 함수의 기본 사용법
  - ggvis() 함수를 사용하려면 **ggvis** 패키지를 불러와야 함
  - 아래 코드는 built-in 데이터셋인 mtcars 의 wt(차량 중량)과 mpg(연비)간의 산점도를 ggvis() 함수를 사용해 작성하는 예임

```
library(ggvis)

p <- ggvis(mtcars, x = ~wt, y = ~mpg)
layer_points(p)
```



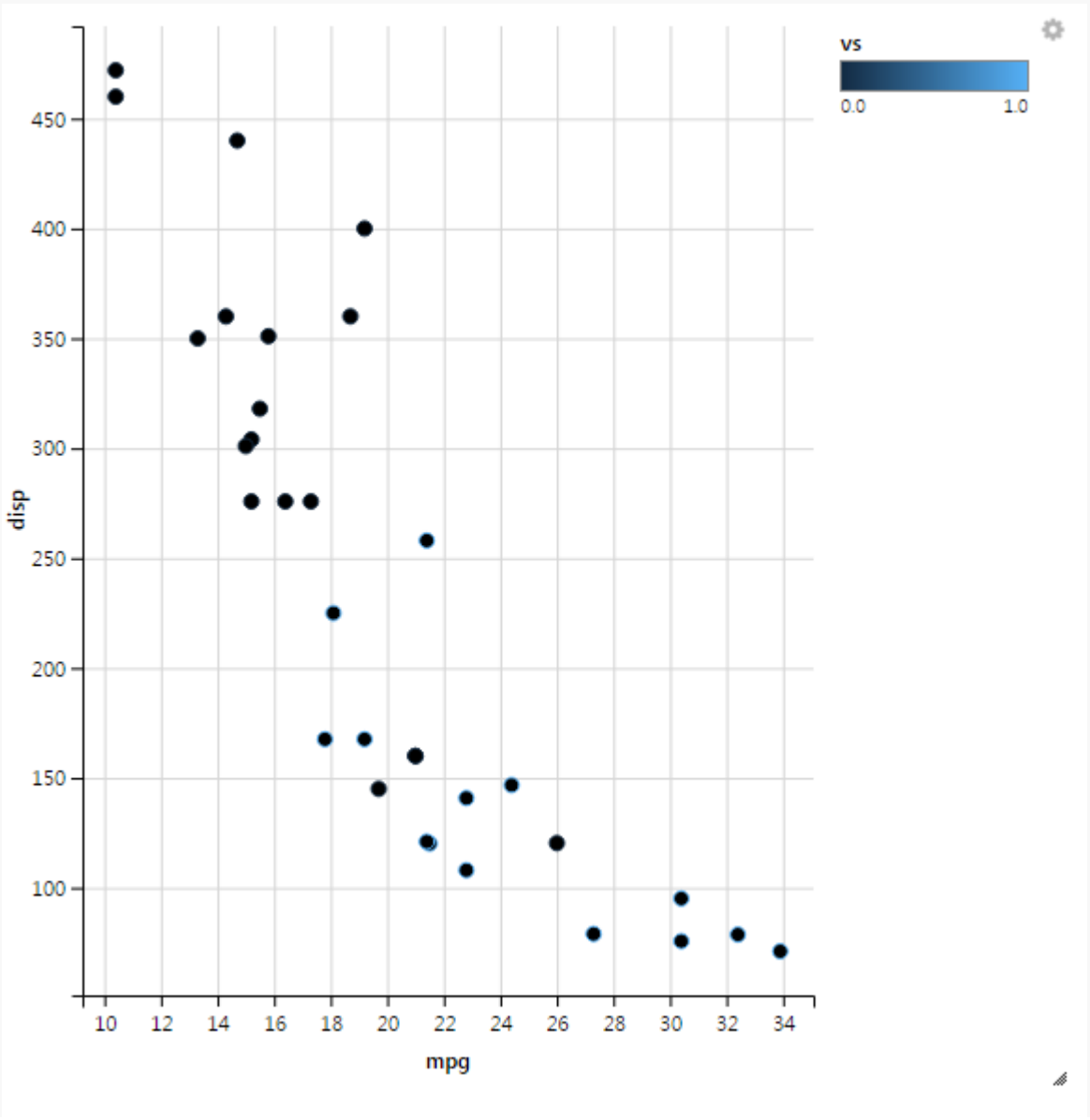
- pipe operator 사용이 가능하기 때문에 보다 직관적이고 깔끔한 코딩이 가능함
  - 아래 코드는 위에서 수행했던 산점도 작성 작업을 파이프 연산자를 사용해 동일하게 수행하는 예임

```
mtcars %>%
  ggvis(x = ~wt, y = ~mpg) %>%
  layer_points()
```



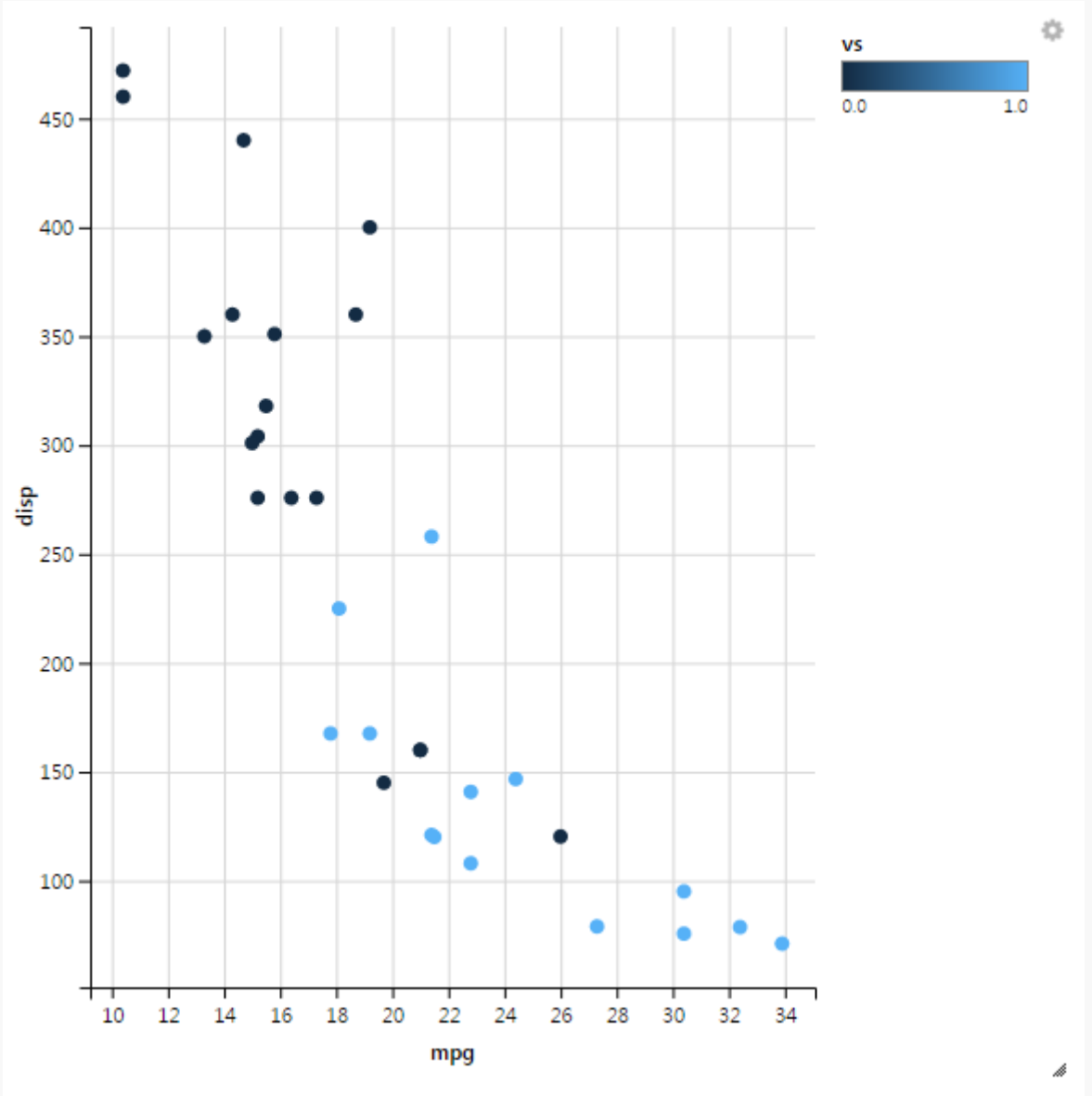
- ggvis() 함수에서 지정 가능한 property
  - stroke =

```
mtcars %>%
  ggvis(~mpg, ~disp, stroke = ~vs) %>%
  layer_points()
```



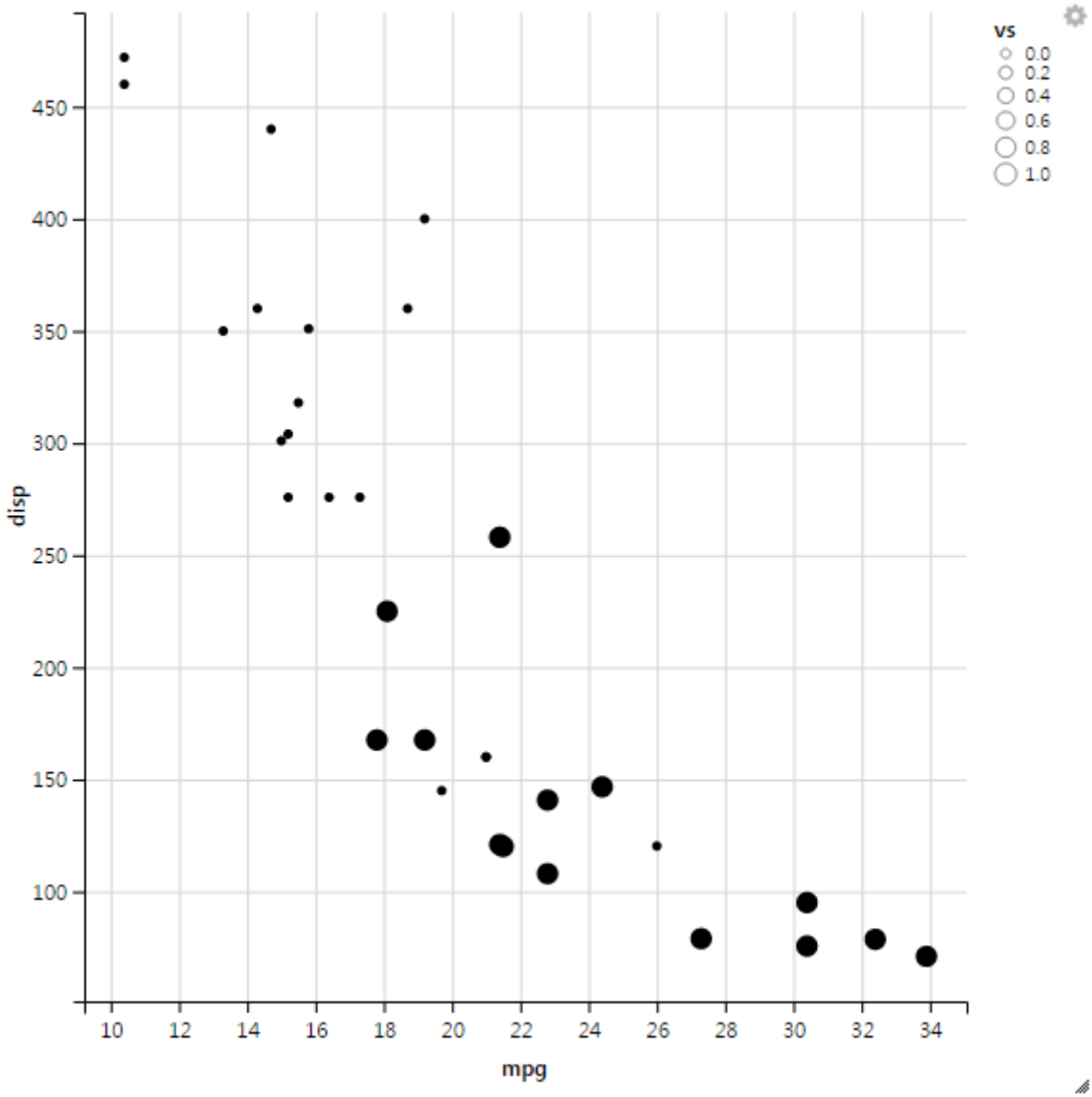
- `fill =`

```
mtcars %>%
  ggvis(~mpg, ~disp, fill = ~vs) %>%
  layer_points()
```



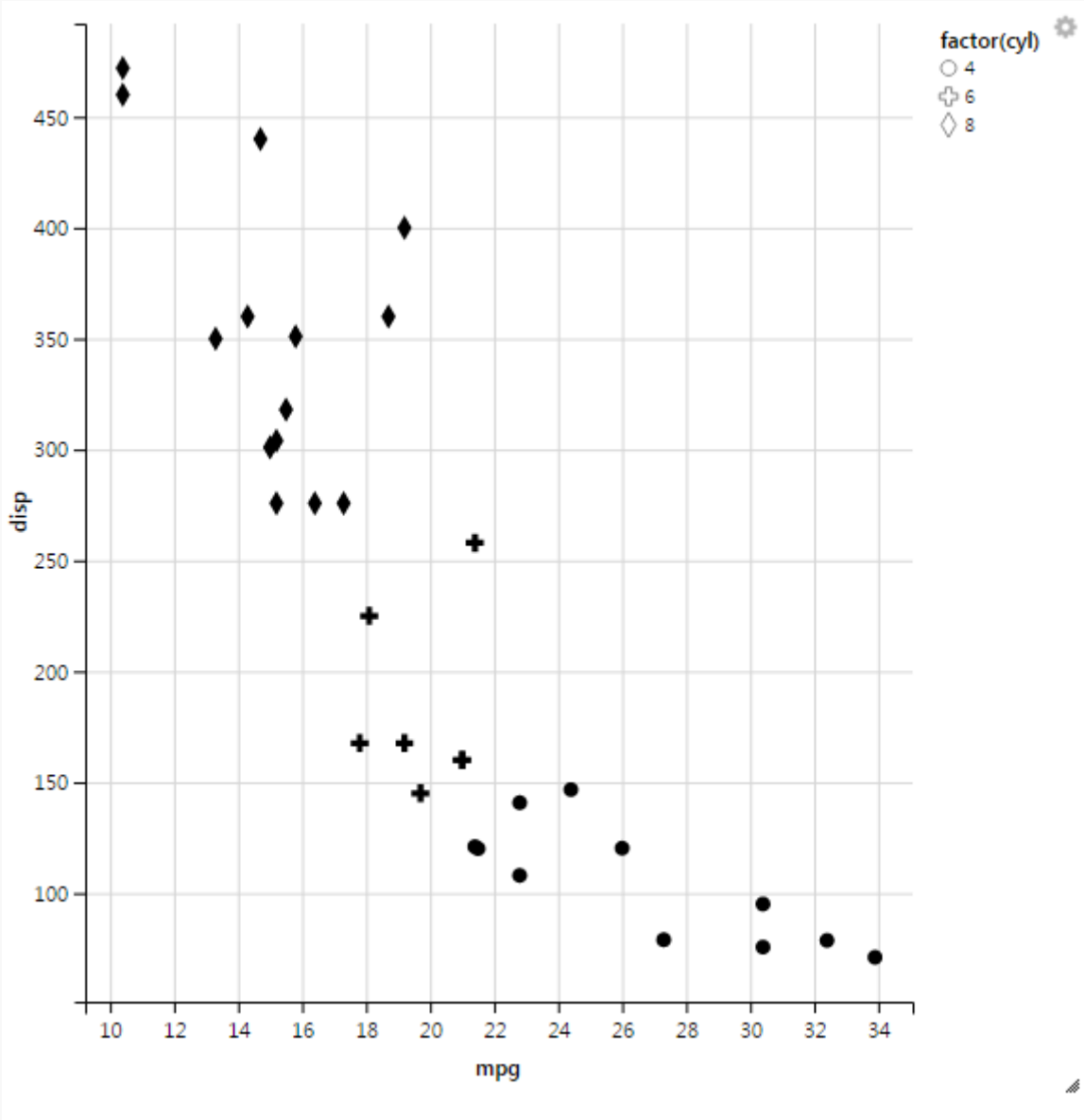
- size =

```
mtcars %>%
  ggvis(~mpg, ~disp, size = ~vs) %>%
  layer_points()
```



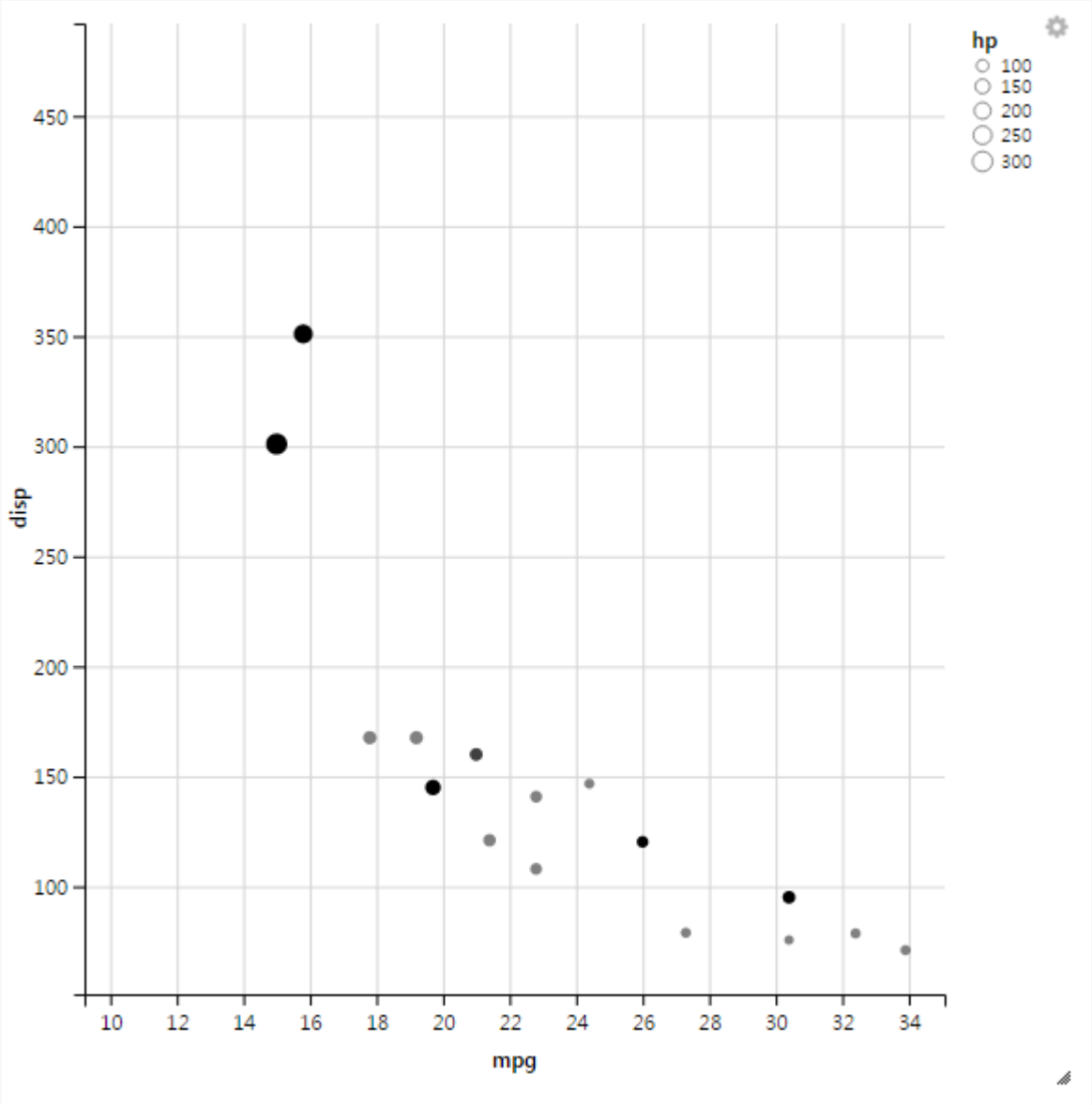
- shape =

```
mtcars %>%
  ggvis(~mpg, ~disp, shape = ~factor(cyl)) %>%
  layer_points()
```



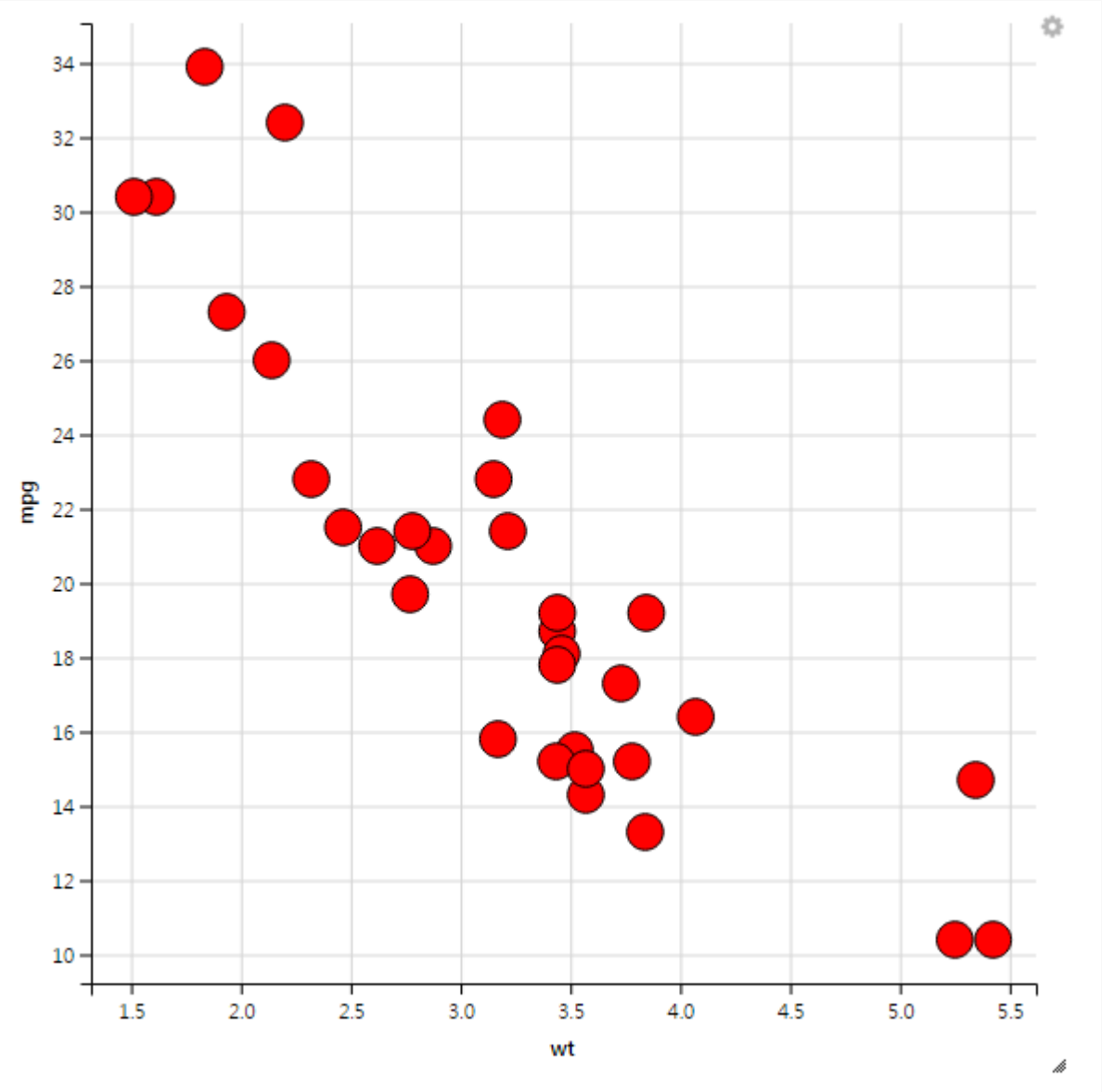
- opacity=

```
mtcars %>%
  ggvis(~mpg, ~disp, size = ~hp, opacity = ~gear) %>%
  layer_points()
```

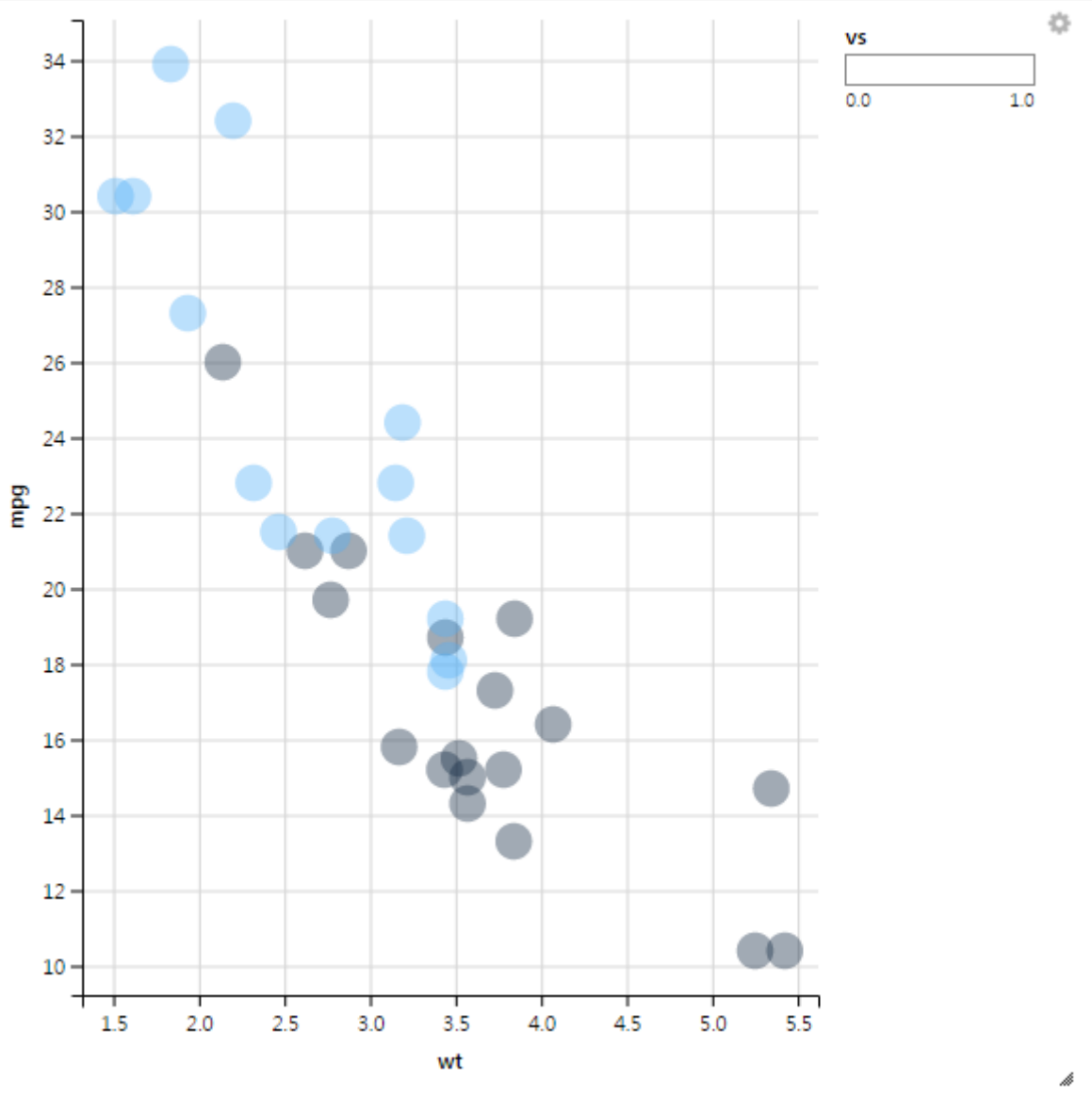


- 고정 property 값
  - property 값을 =로 지정하면 ggvis() 함수가 지정된 변수의 값을 scale 해 자동으로 property 값을 결정함
  - :=을 사용하면 지정된 값으로 property 값을 정함

```
mtcars %>%
  ggvis(~wt, ~mpg, size := 300, fill := "red", stroke := "black") %>%

  layer_points()
```



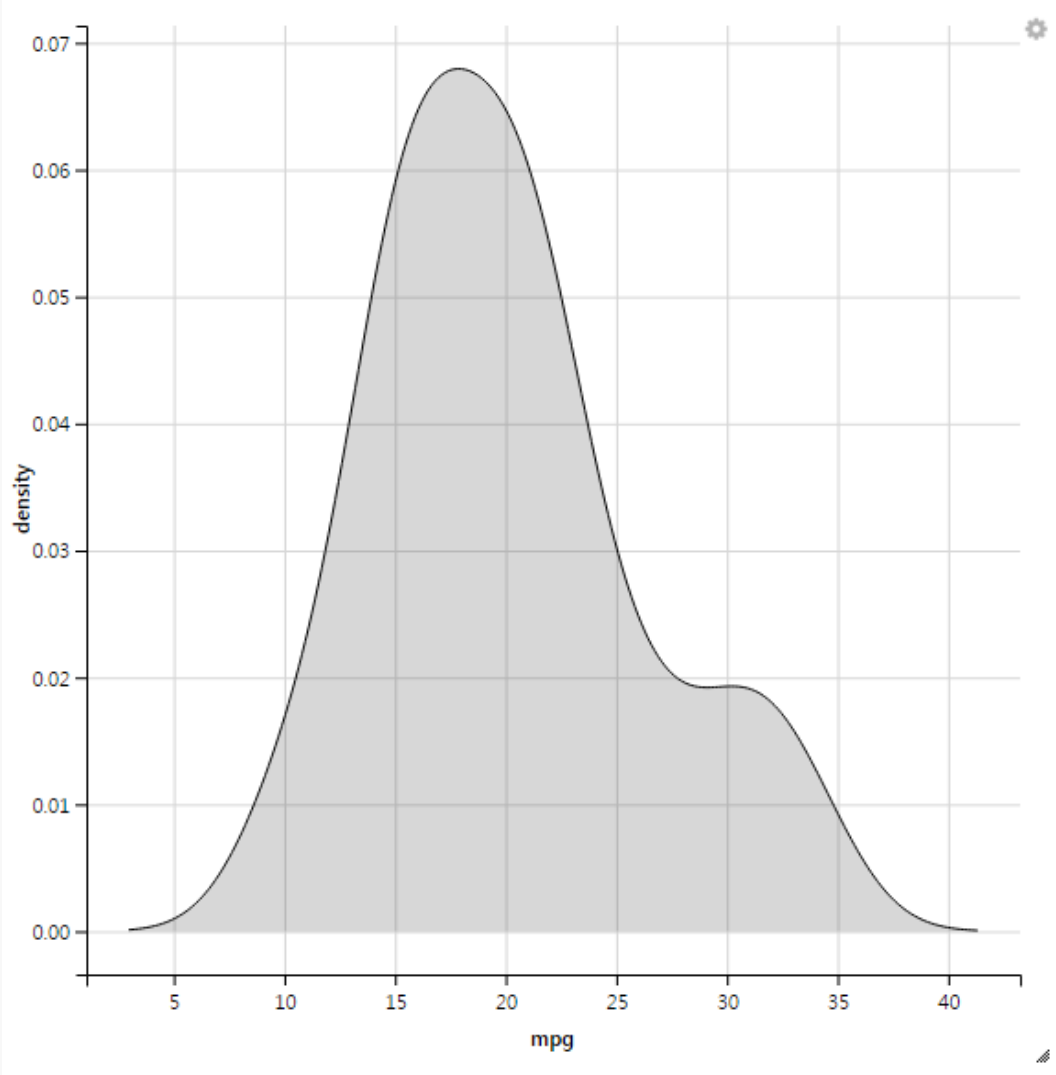
```
mtcars %>%
  ggvis(~wt, ~mpg, size := 300, opacity := 0.4, fill = ~vs) %>%
  layer_points()
```



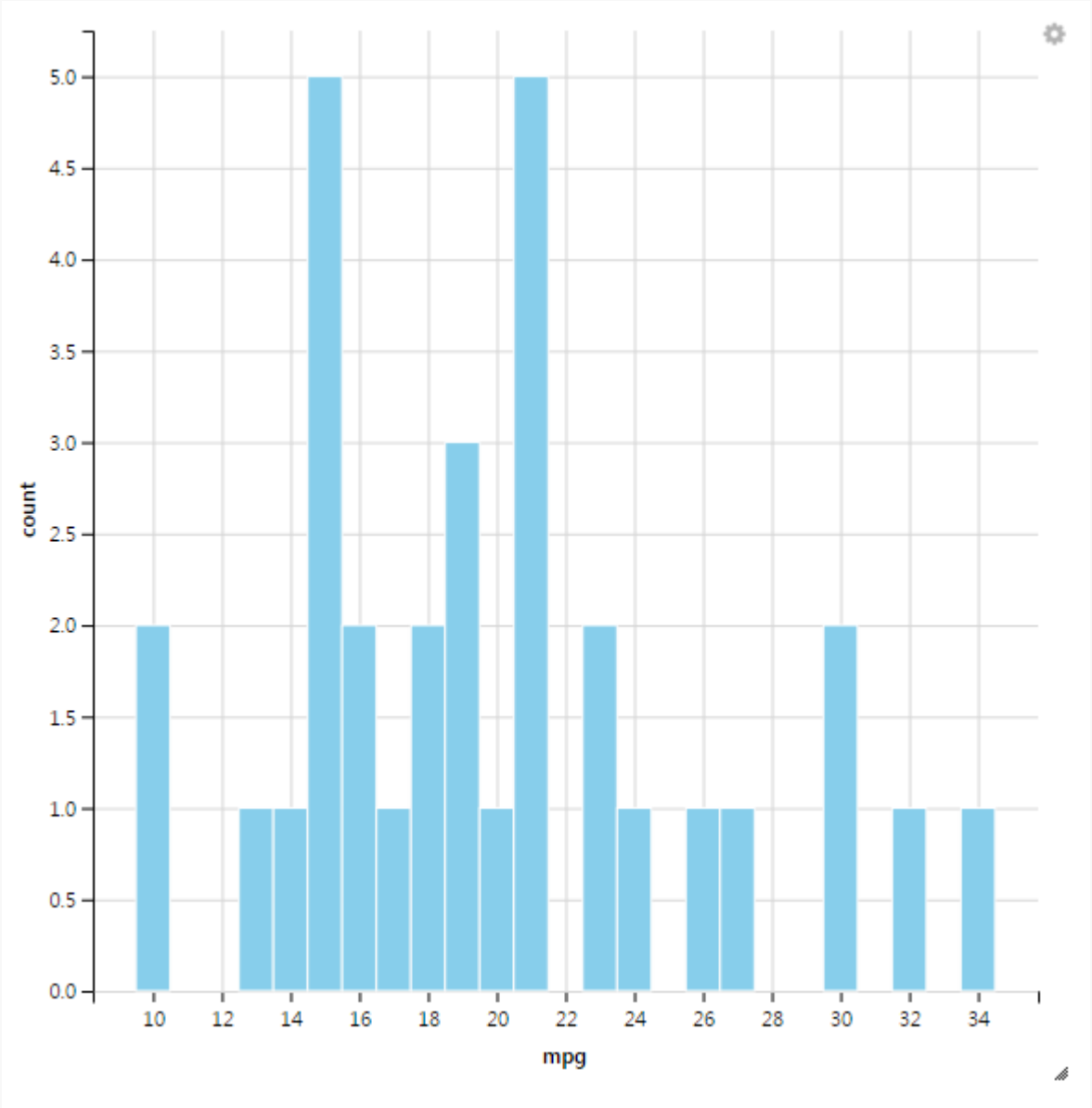
- layer_ 함수
  - 기본 레이어
    - layer_paths()
    - layer_points()
    - layer_rects()
    - layer_ribbons()
    - layer_text()

- 자료분석 레이어
  - layer_arcs()
  - layer_bars()
  - layer_boxplots()
  - layer_densities()
  - layer_freqpolys()
  - layer_histograms()
  - layer_images()
  - layer_model_predictions()
  - layer_smooths()

```
mtcars %>%
  ggvis(~mpg) %>%
  layer_densities()
```

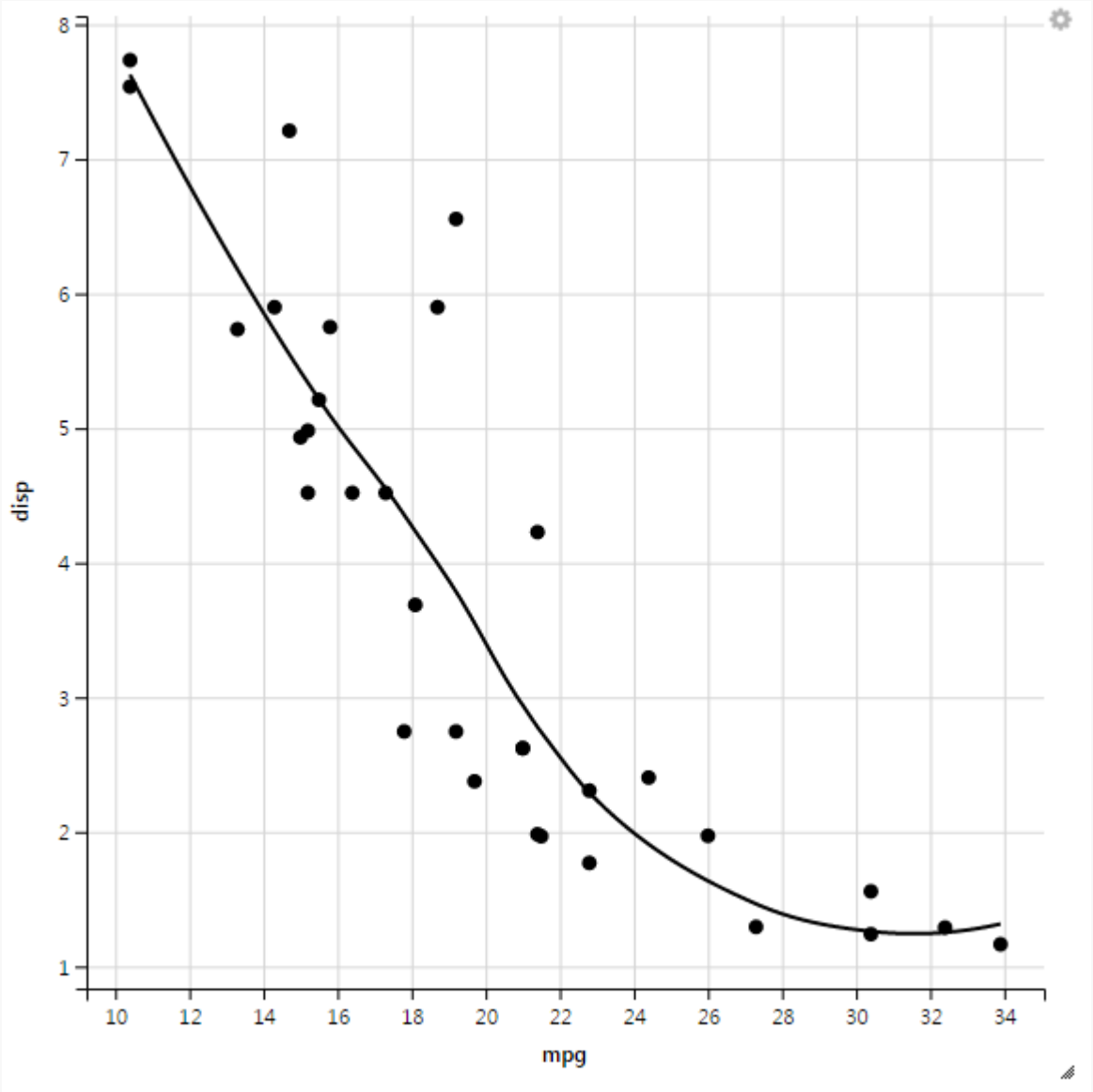


```
mtcars %>%
  ggvis(~mpg, fill := "skyblue", stroke := "white") %>%
  layer_histograms()
```



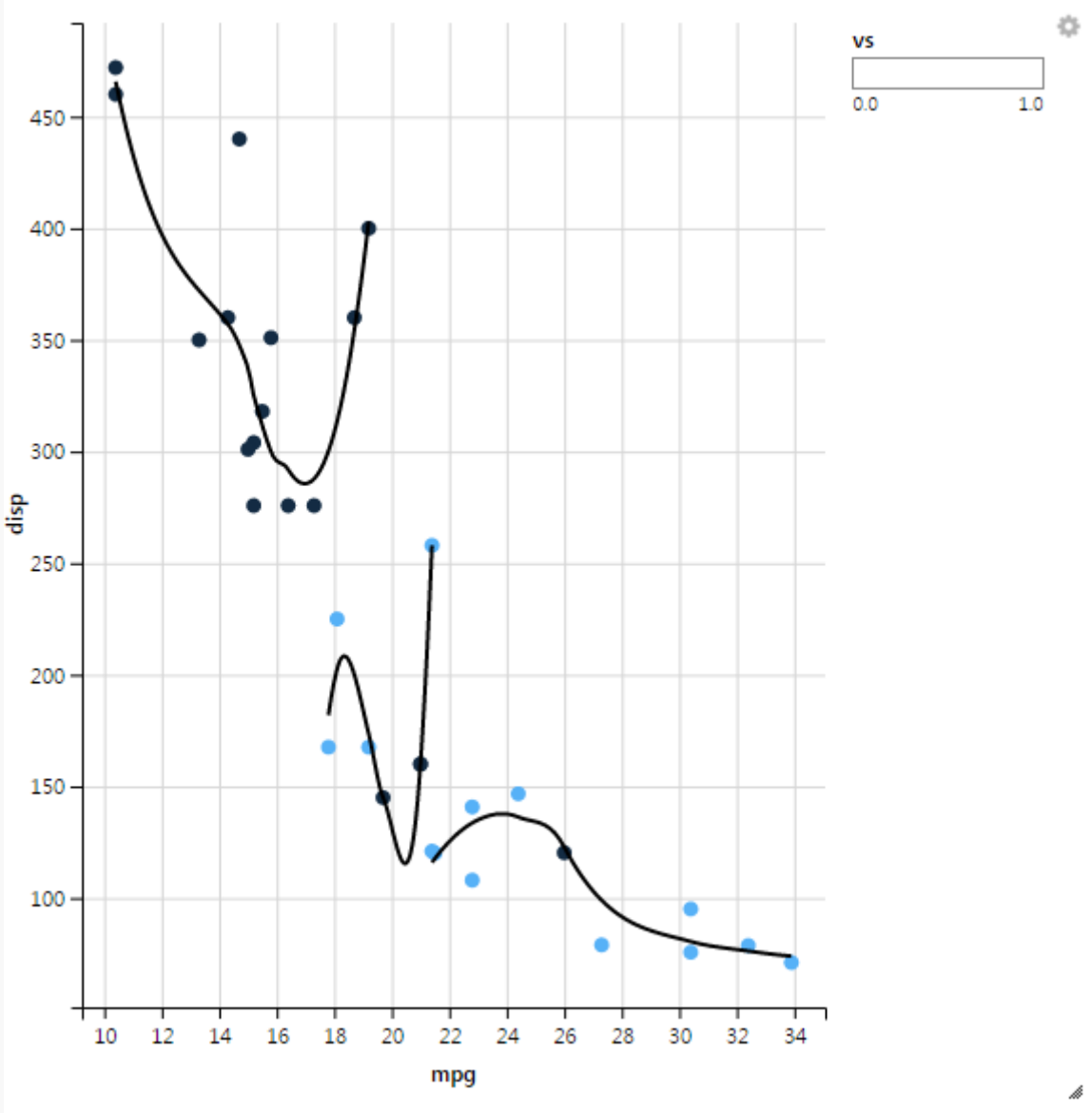
- 레이어 겹치기
  - layer_함수를 파이프를 사용해 연속해 사용하면 겹쳐진 그래프 작성

```
library(dplyr)
mtcars %>%
  ggvis(x = ~mpg, y = ~disp) %>%
  mutate(disp = disp / 61.0237) %>% # convert engine displacment to l
itres
  layer_points() %>%
  layer_smooths()
```

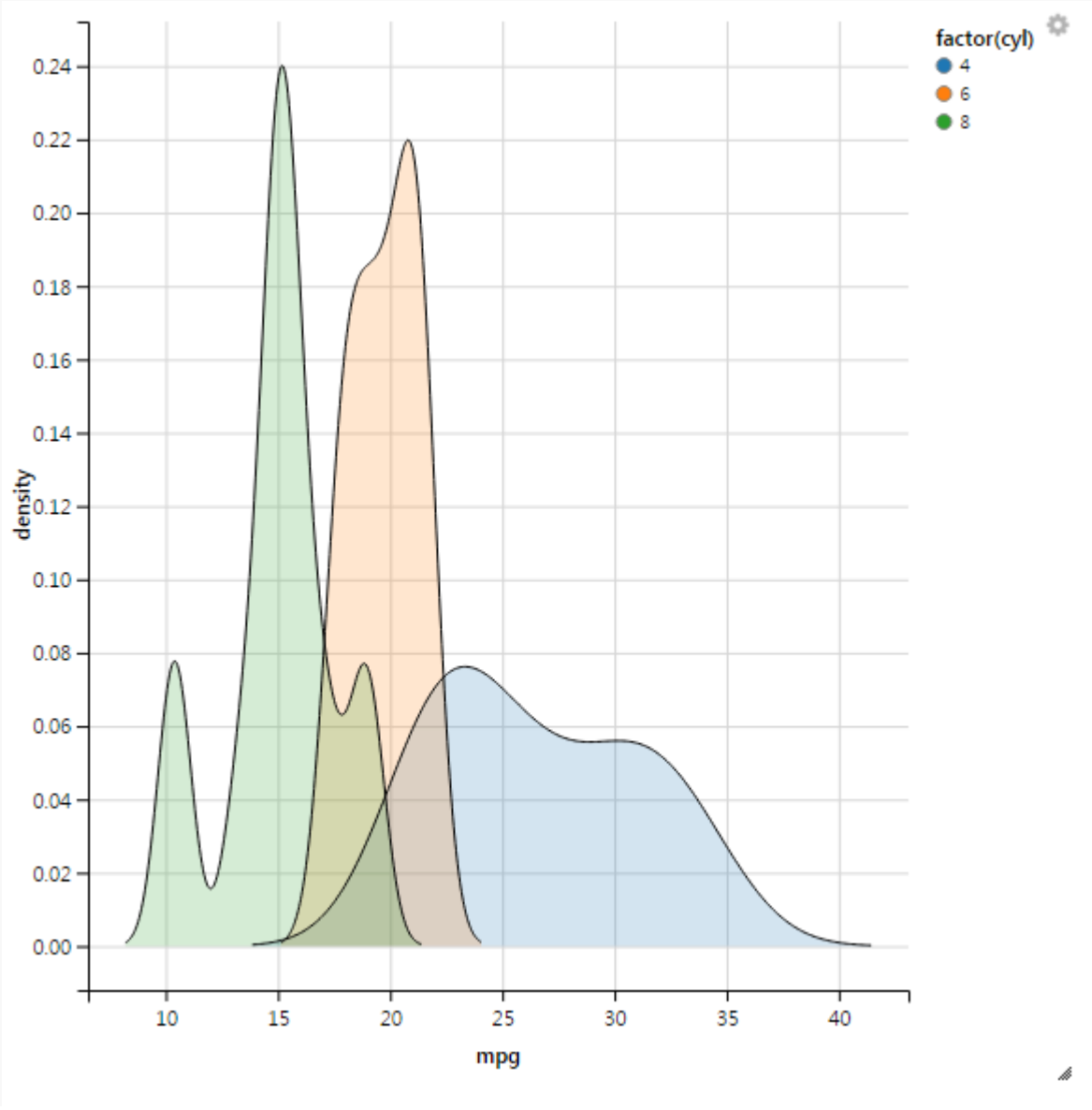


- 그룹데이터
  - 그룹 변수가 있는 경우 그룹별 작업이 가능
  - **dplyr** 패키지의 group_by() 함수 이용

```
mtcars %>%
  group_by(cyl) %>%
  ggvis(~mpg, ~disp, fill = ~vs) %>%
  layer_points() %>%
  layer_smooths()
```



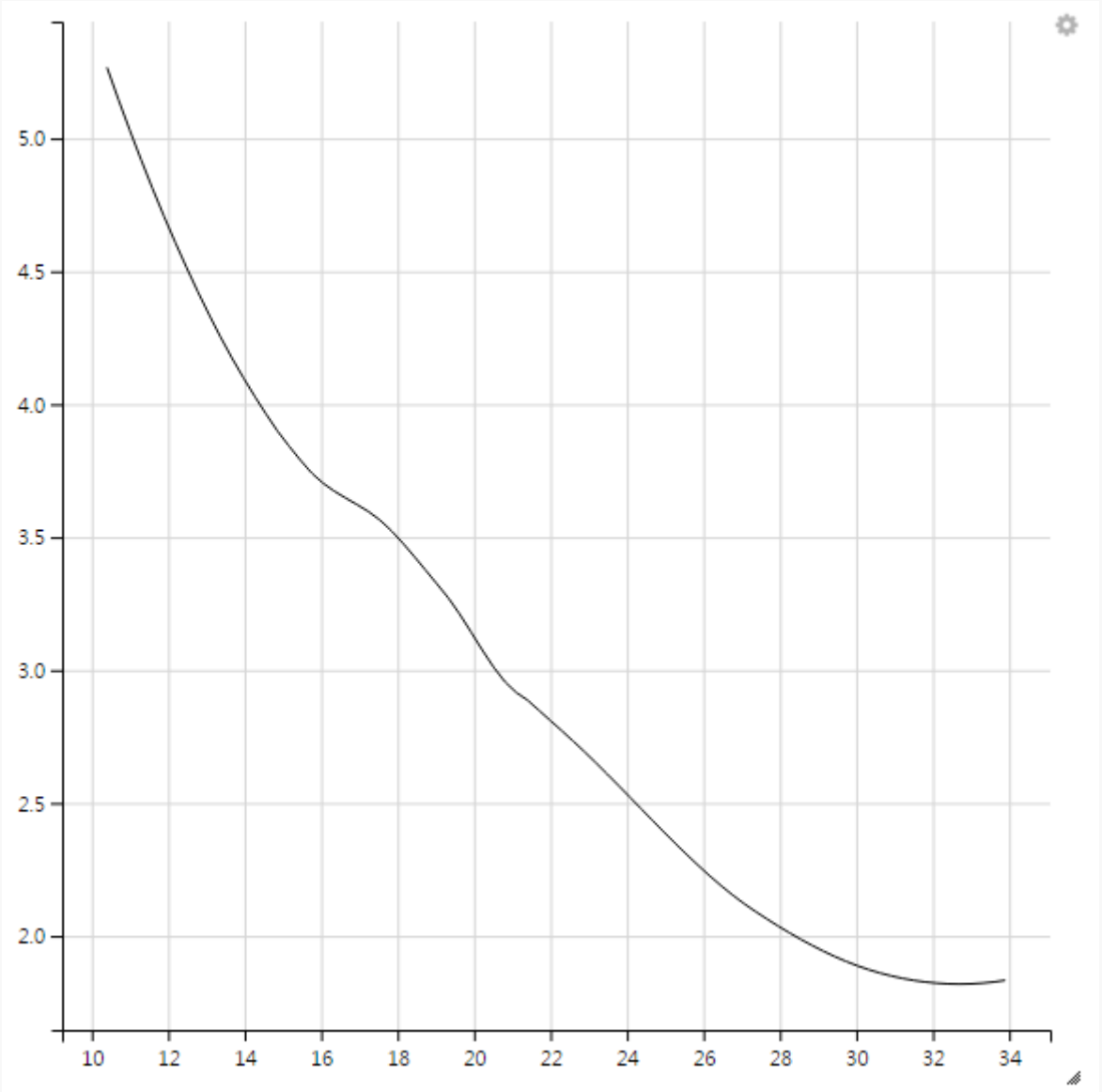
```
mtcars %>%
  group_by(cyl) %>%
  ggvis(~mpg, fill = ~factor(cyl)) %>%
  layer_densities()
```



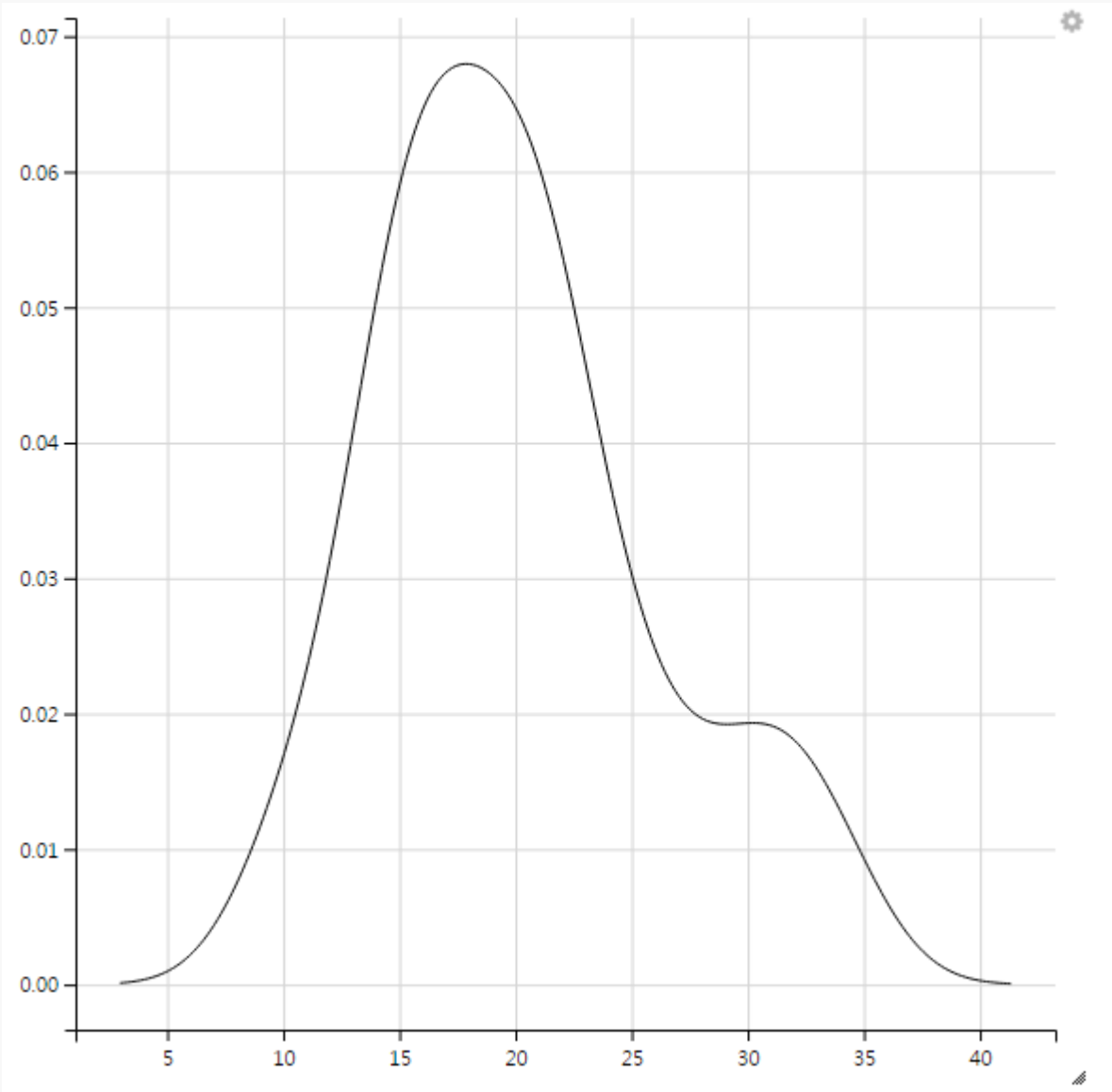
- 간단한 분석을 해주는 compute_ 함수
  - compute_align()
  - compute_bin()
  - compute_boxplot()
  - compute_count()
  - compute_density()
  - compute_model_predictions()

- compute_smooth()
- compute_stack()
- compute_tabulate()

```
mtcars %>%
  compute_smooth(wt ~ mpg)

mtcars %>%
  compute_smooth(wt ~ mpg) %>%
  ggvis(~pred_, ~resp_) %>%
  layer_paths()
```



```
mtcars %>%
  compute_density(~ mpg) %>%
  ggvis(~pred_, ~resp_) %>%
  layer_paths()
```

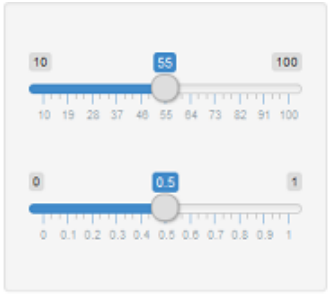


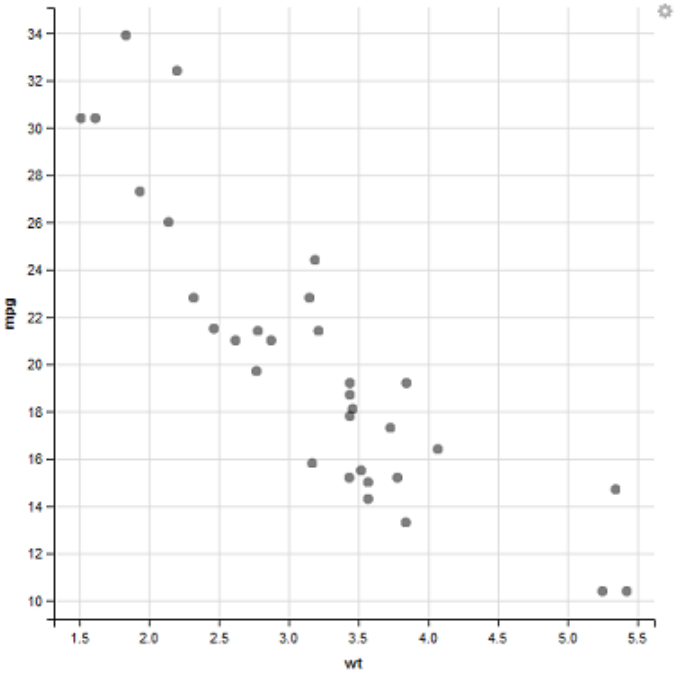
B.2 인터랙티브 그래픽

- 그래픽에 대한 인터랙티브한 제어가 가능
- 정적인(static) 그래픽이 아니기 때문에 보고서 작성 및 publish 에 사용하는 것이 불편
- `input_` 함수 사용
  - `input_checkbox()`
  - `input_checkboxgroup()`
  - `input_numeric()`
  - `input_radiobuttons()`
  - `input_select()`
  - `input_slider()`
  - `input_text()`
- 아래 코드는 산점도 내 점의 크기와 투명도를 슬라이드 방식으로 제어할 수 있는 인터랙티브 그래픽을 작성하는 예임

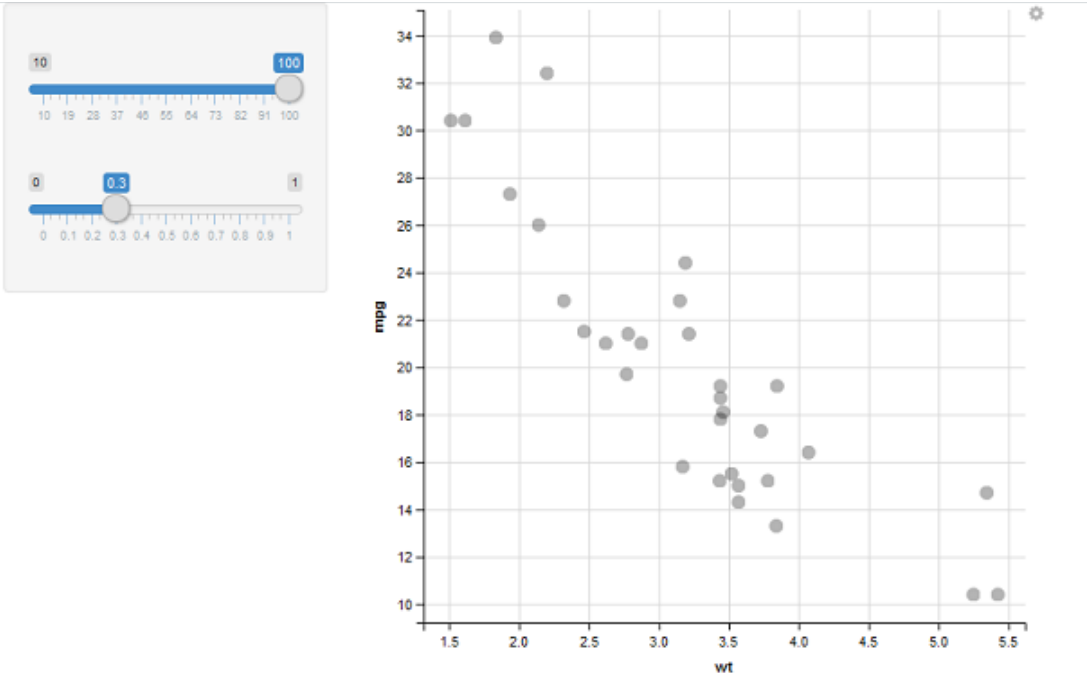
```
mtcars %>%
  ggvis(~wt, ~mpg,
    size := input_slider(10, 100),
    opacity := input_slider(0, 1)
  ) %>%
  layer_points()
```

  - 점 크기 size = 55, 투명도 opacity = 0.5 인 경우 화면 캡쳐





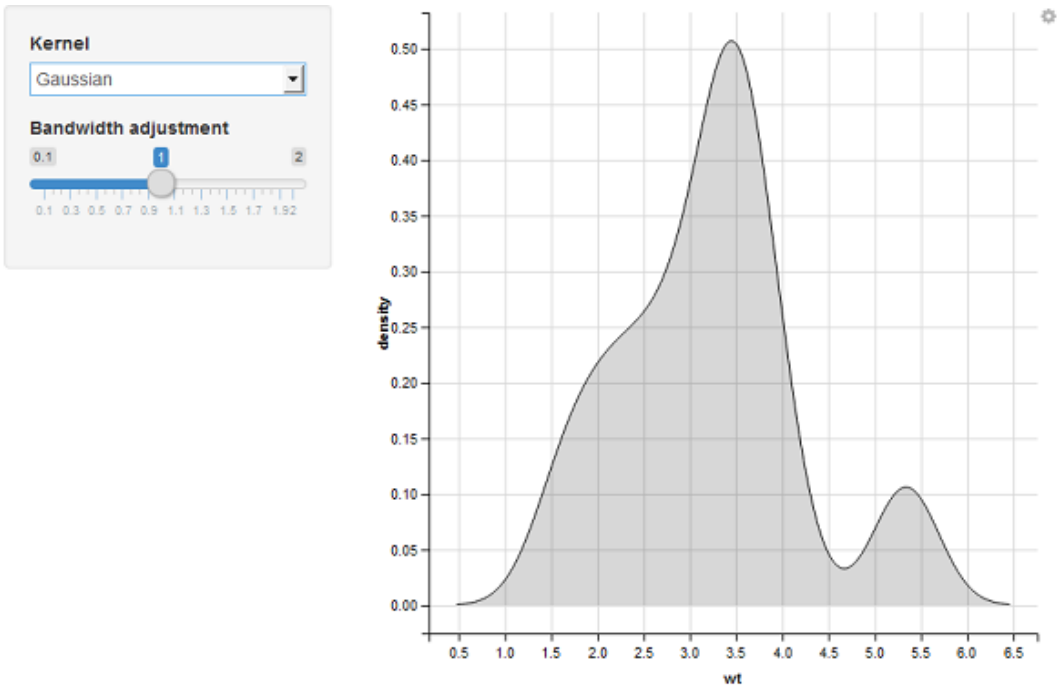
- 점 크기 size = 100, 투명도 opacity = 0.3 인 경우 화면 캡쳐



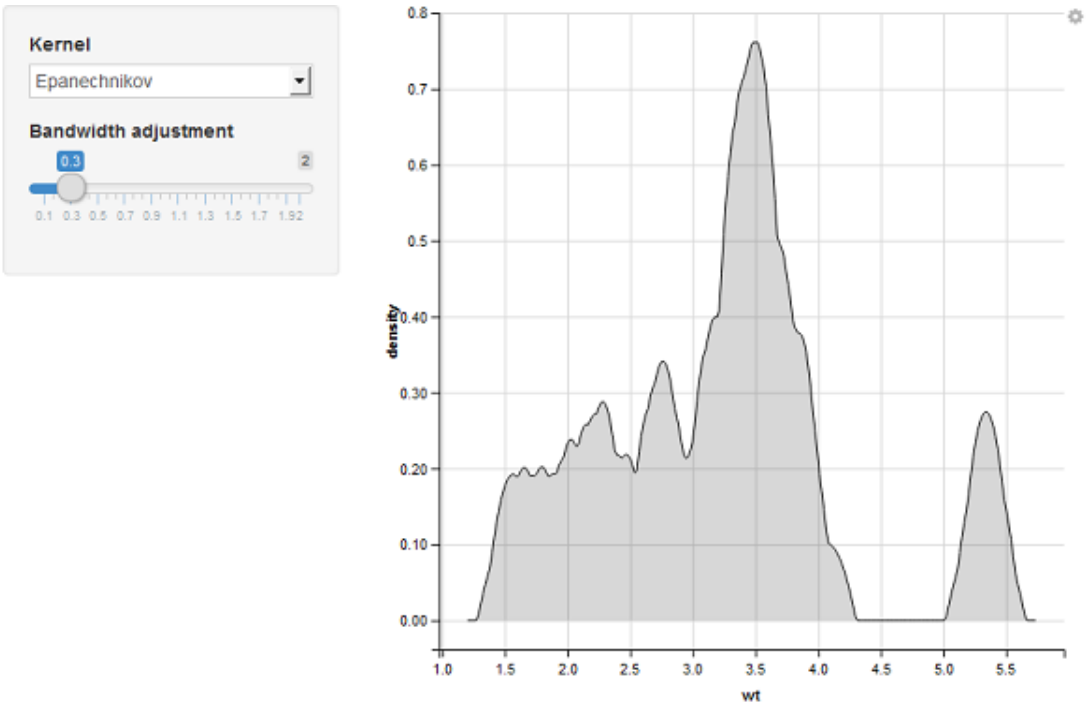
- 아래 코드는 커널밀도함수추정에 사용할 커널과 bandwidth 를 인터랙티브하게 선택할 수 있는 그래픽 작성의 예임

```
    sliderBox <- input_slider(.1, 2, value = 1, step = .1, label = "Bandw
idth adjustment")
    selectBox <- input_select(c("Gaussian" = "gaussian",
                                "Epanechnikov" = "epanechnikov",
                                "Rectangular" = "rectangular",
                                "Triangular" = "triangular",
                                "Biweight" = "biweight",
                                "Cosine" = "cosine",
                                "Optcosine" = "optcosine"
                                ),
                              label = "Kernel")
    mtcars %>%
      ggvis(x = ~wt) %>%
      layer_densities(adjust = sliderBox, kernel = selectBox)
```

- Kernel = Gaussian, Bandwidth = 1 인 경우 화면 캡쳐



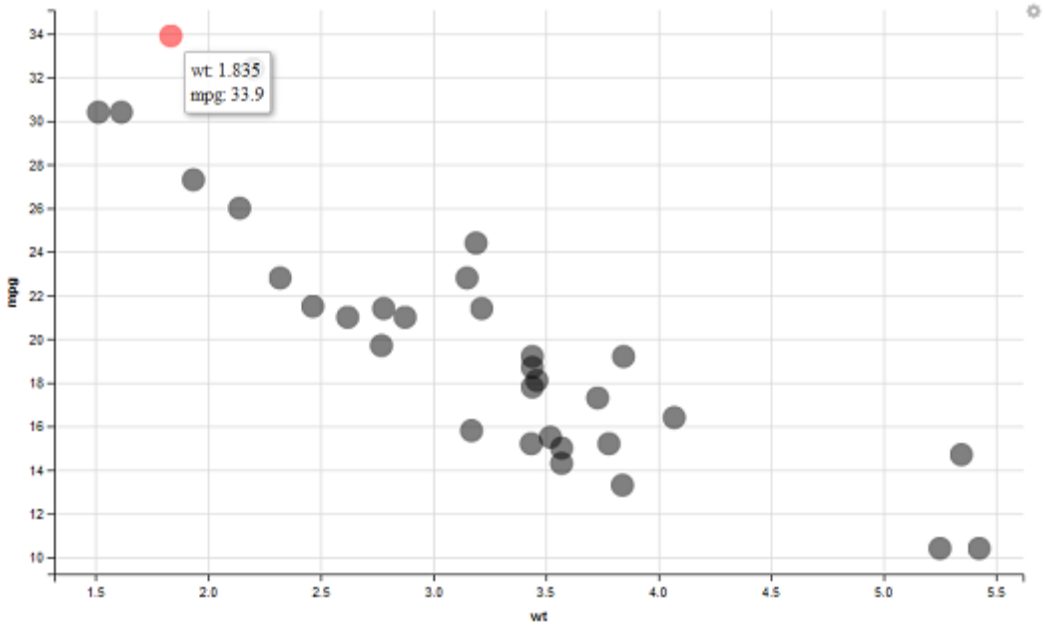
- Kernel = Epanechnikov, Bandwidth = 0.3 인 경우 화면 캡쳐



- tooltip 이라 불리는 보다 복잡한 인터랙티브 작업 가능
  - 아래 코드는 산점도 내의 점 위에 마우스를 갖다대면(hover) 점의 색이 달라지면서 해당 점에 대한 정보를 보여주는 예임

```
all_values <- function(x)
{
  if(is.null(x)) return(NULL)
  paste0(names(x), ": ", format(x), collapse="<br />")
}

mtcars %>%
  ggvis(~wt, ~mpg, size := 300, opacity := 0.5) %>%
  layer_points(fill.hover := "red") %>%
  add_tooltip(all_values, "hover")
```

- 산점도의 맨 위에 있는 점 위에 마우스를 대었을 때 화면 캡쳐



C. 범주형 자료분석 기법

C.1 분할표 분석

분할표(contingency table) 분석은 범주형 변수 간의 연관성 분석을 위해 대표적으로 사용되는 방법이다. 데이터를 두 요인에 의해 분할해 도수를 조사하고 분할표를 작성하여, 분할에 사용된 요인 사이에 관련이 있는 지 또는 독립인 지 여부를 검정하게 된다.

아래는 신용카드 사용 데이터로부터 한화(KRW)로 결제한 경우 중 시험용 데이터를 제외한 데이터를 골라낸 후, 사용 부문(CATEGORY_TYPE)과 카드사(PAY_CD) 간에 분할표를 작성하는 코드이다.
분할표 작성에는 `table()` 함수를 이용한다.

```
    card.dat <- card.df %>%
      filter(FOREIGN_UNIT == "KRW",
             TRADE_SITE_NM != "테스트" |
               TRADE_SITE_NM != "시험 1" |
               TRADE_SITE_NM != "시험 2" |
               TRADE_SITE_NM != "시험 3" |
               TRADE_SITE_NM != "시험 4")
    mytable <- with(card.dat, table(PAY_CD, CATEGORY_TYPE))
    mytable
```

```
##       CATEGORY_TYPE
## PAY_CD          CA    CB    CC    CD    CE    CF    CG    CH    CI    CJ
##           0     0     0     0     0     0     0     0     0     0     0
##     AM    0  4473   982   590   466   504    84   604   168    41   154
##     BC    0 25573  8464  4179  2347  4307   642  8736   922   548  1084
##     BK    0    56     2     1     0     8     7     7     4    10    60
##     DI    0 21484  7510  3060  3433  2600   596  6720   554   171  1365
##     KE    0  5222  1782  1789   607   357   217  1625   223    46   165
##     KM    0 24477  7384  3783  1594  3111   883  6579  2586   568   884
##     SH    0 34185 11059  4065  2730  3596   869  9715   651   221  1211
##     SS    0 10485  3477  1211  1047  1327   277  2503   488    51   627
##       CATEGORY_TYPE
## PAY_CD    CK    CL    CZ
##           0     0     0
##     AM    0     0  2455
##     BC    1   255 23686
##     BK    0     0   341
##     DI    0    13  9925
##     KE    0   182  8149
##     KM    0   263 12396
##     SH   50   193 26383
##     SS    0     6  6212
```

두 요인이 **독립(independent)**이라는 가설 하에 구한 셀 도수의 기대값과 관측된 셀 도수 간의 차이를 계산해 **카이제곱 검정(Chi-square test)**을 실시한다.
이 때 귀무가설은 'H_0: 두 요인이 서로 무관하다'가 된다.
카이제곱검정 결과 계산된 p-값이 미리 정한 유의수준 α보다 작으면 귀무가설을 기각한다. 즉, 두 요인 간에 서로 연관성이 있다고 결론을 내린다.

보통 의사 결정의 기준치인 유의수준 α의 값은 0.05 을 사용하며, 보수적으로 의사 결정을 해야 하는 경우 0.01 을 사용하기도 한다.

위에서 작성한 분할표를 살펴보면 사용 부문이 CK, CL 인 경우와 카드사가 BK 인 경우 분할표의 빈 셀(cell)이 다수 발생한 것을 알 수 있다.
카이제곱 검정이 잘 작동할 수 있도록 해당 케이스는 제외하고 분석을 진행하자.

```
    chisq.test(mytable[-c(1, 4), -c(1, 12, 13)])
```

```
##
##  Pearson's Chi-squared test
##
## data:  mytable[-c(1, 4), -c(1, 12, 13)]
## X-squared = 14383, df = 60, p-value < 2.2e-16
```

- p-값이 지극히 작은 값임을 확인
- 따라서, 유의수준 $\alpha = 0.05$하에서 귀무가설 'H_0: 사용 부문과 카드사는 무관하다'를 기각
- 즉, 사용 부문과 카드사 간에 관련성이 있다고 할 근거가 있음

위의 분석 결과는 카드사별로 잦은 매출이 발생하는 사용 부문이 다를 수 있음을 시사한다.
그러나 카드사별 특화된 사용 부문을 분할표를 이용해 직접 따져보기에는 분할표의 크기가 너무 커서 한 눈에 알아보기에는 어려움이 있다.
보다 심화된 분석을 위해 고급 분석 기법이 필요하다.

C.2 대응일치분석

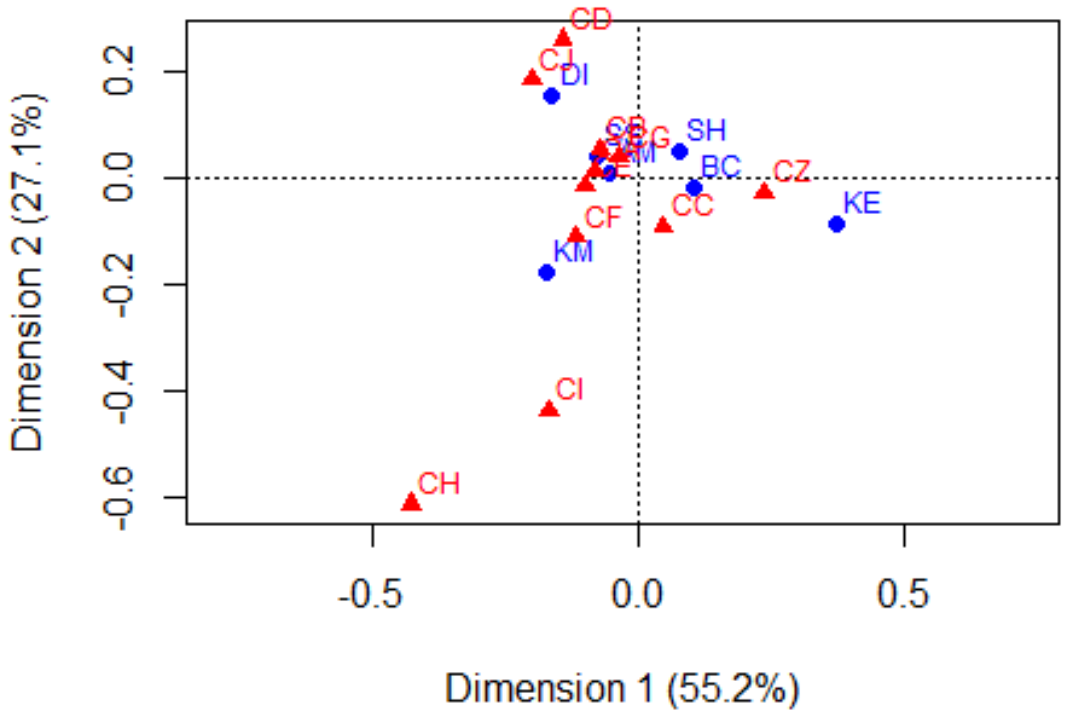
대응일치분석(correspondence analysis, CA)은 범주형 변수 간의 연관성을 해석 가능한 방식으로 시각적으로 확인하는데 효과적인 분석 방법이다.
두 개의 범주형 변수 간의 연관성 분석에 사용하며(세 개 이상은 다중대응일치분석 MCA) R 의 **ca** 패키지를 이용하면 편리하다.

소비 데이터의 지출 카테고리(CATEGORY_TYPE)와 신용카드회사(PAY_CD) 간의 이원분할표(two-way contingency table)를 기반으로 대응일치분석을 실시해 보자.
아래 코드는 ca 패키지를 불러온 후, ca() 함수를 사용해 위에서 작성한 분할표에 대한 대응일치분석을 실시한 결과를 ca.fit 라는 이름의 객체로 저장하고, 이를 시각화하는 예이다.

```
    library(ca)

    ca.fit <- ca(mytable[-c(1, 4), -c(1, 12, 13)])
    summary(ca.fit)
```

```
##
## Principal inertias (eigenvalues):
##
##  dim    value      %   cum%   scree plot
##  1      0.022335  55.2  55.2  **************
##  2      0.010992  27.1  82.3  *******
##  3      0.003316   8.2  90.5  **
##  4      0.002416   6.0  96.5  *
##  5      0.001052   2.6  99.1  *
##  6      0.000378   0.9 100.0
##         -------- -----
##  Total: 0.040489 100.0
##
##
## Rows:
##     name   mass  qlt  inr    k=1 cor ctr    k=2 cor ctr
## 1 |   AM |   30   70   32 |  -54  68   4 |    9   2   0 |
## 2 |   BC |  227  634  100 |  105 617 111 |  -18  17   6 |
## 3 |   DI |  162  909  224 | -163 475 193 |  156 433 357 |
## 4 |   KE |   57  826  248 |  372 782 352 |  -88  44  40 |
## 5 |   KM |  181  996  280 | -174 484 246 | -179 512 529 |
## 6 |   SH |  267  621   89 |   78 449  72 |   48 172  56 |
## 7 |   SS |   78  540   28 |  -78 428  22 |   40 112  11 |
##
## Columns:
##      name   mass  qlt  inr    k=1 cor ctr    k=2 cor ctr
## 1  |   CA |  354  717   85 |  -83 702 108 |   12  15   5 |
## 2  |   CB |  114  797   27 |  -70 516  25 |   52 280  28 |
## 3  |   CC |   53  271   53 |   48  57   5 |  -93 214  42 |
## 4  |   CD |   34  740  101 | -143 173  31 |  259 567 210 |
## 5  |   CE |   44  268   41 |  -99 260  20 |  -17   8   1 |
## 6  |   CF |   10  691    9 | -116 355   6 | -113 335  12 |
## 7  |   CG |  103  202   34 |  -36  97   6 |   38 106  13 |
## 8  |   CH |   16  986  222 | -429 323 130 | -615 664 542 |
## 9  |   CI |    5  673   38 | -168  86   6 | -439 587  81 |
## 10 |   CJ |   15  821   34 | -201 449  28 |  183 372  47 |
## 11 |   CZ |  251  998  356 |  238 984 634 |  -29  14  19 |
```

```
plot(ca.fit)
```



- 대응일치분석을 실시한 결과 구축된 2 개 차원의 설명력이 82.3%로서(Principal inertias: cum%값과 scree plot 참조) **기준치인 70%**를 상회
  - 분석 결과에 의한 사후 분석을 진행해도 무방함
- 대응일치분석 실시 결과 그림
  - 그림에 찍힌 빨간색 점(카테고리)과 파란색 점(카드사) 간의 거리가 가까울수록 카드사 매출이 해당 카테고리에서 많이 발생했음을 의미함
  - 따라서 이 그림을 활용하면 시장 내에서 각 카드사의 시장 내 포지셔닝이 어떠한 지 분석 가능함
  - 카드사 중 AM 사, SS 사는 CA(마트/쇼핑), CB(외식/부식), CE(건강/의료), CG(교통/주유) 등 고른 부문에서 매출이 자주 발생

- SH 사와 BC 사는 CA(마트/쇼핑), CB(외식/부식), CE(건강/의료) 등에서도 매출이 많이 발생하지만, CG(교통/주유), CC(커피/간식) 부문에 특화된 것을 확인
- KM 사의 경우 타 사에 비해 CF(미용/뷰티), CI(교육/학원), CH(주거/생활) 부문에서 매출이 많이 발생. 주부 층의 사용이 많은 것으로 예상됨
- DI 사의 경우 타 사에 비해 CJ(보험/세금), CD(레저/문화) 부문에 특화된 것을 확인할 수 있음. 남성 고객이 많은 것으로 예상됨
- KE 사는 시장 내 포지션이 애매하며 특화된 부문이 보이지 않음