관계형데이터

Table of Contents

[관계형 데이터 2](#_Toc490834918)

[관계형 데이터란 2](#_Toc490834919)

[관계형 데이터의 작업 2](#_Toc490834920)

[사용하는 데이터 2](#_Toc490834921)

[데이터의 관계 3](#_Toc490834922)

[키(keys) 4](#_Toc490834923)

[키의 종류 4](#_Toc490834924)

[기본키의 확인 5](#_Toc490834925)

[대체키(surrogate key) 5](#_Toc490834926)

[관계(relation) 6](#_Toc490834927)

[연습문제 6](#_Toc490834928)

[변수 추가(Mutating joins) 8](#_Toc490834929)

[Mutating joins 8](#_Toc490834930)

[join의 이해 9](#_Toc490834931)

[Inner join 10](#_Toc490834932)

[Outer join 11](#_Toc490834933)

[중복되는 키(Duplicate keys) 12](#_Toc490834934)

[키 열(들)을 정의하기 14](#_Toc490834935)

[연습문제 16](#_Toc490834936)

[base R 과의 비교 24](#_Toc490834937)

[SQL 과의 비교 25](#_Toc490834938)

[필터링 조인(Filtering joins) 25](#_Toc490834939)

[연습문제 28](#_Toc490834940)

## 관계형 데이터

### 관계형 데이터란

데이터 분석을 할 때 하나의 데이터로만 작업하는 경우는 매우 드물며 보통 두 개 이상의 데이터를 조합하여 분석하는 경우가 많다. 서로 관계가 있는 두 개 이상의 데이터는 관계형 데이터라고 할 수 있다.

### 관계형 데이터의 작업

1.변수 추가(Mutating joins): 하나의 데이터프레임에 매칭되는 다른 데이터 프레임의 변수를 추가

2.필터링 조인(Filtering joins): 다른 데이터 프레임에 매칭되는 관찰치 있는지 없는지에 따라 관찰치 필터링

3.집합연산(Set oprerations): 합집합, 교집합, 차집합 등

### 사용하는 데이터

library(tidyverse)  
library(nycflights13)

nycflights13 패키지에는 flights 데이터와 관련된 네개의 데이터가 있다.

* airlines
* airports
* planes
* weather

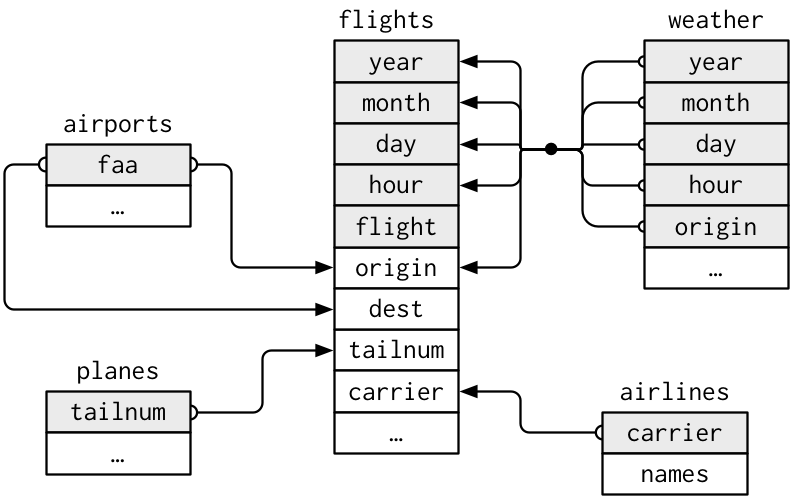
nycflights13에 포함되어 있는 데이터의 이름을 쓰고 실행시켜보자

airlines

# A tibble: 16 x 2  
 carrier name  
 <chr> <chr>  
 1 9E Endeavor Air Inc.  
 2 AA American Airlines Inc.  
 3 AS Alaska Airlines Inc.  
 4 B6 JetBlue Airways  
 5 DL Delta Air Lines Inc.  
 6 EV ExpressJet Airlines Inc.  
 7 F9 Frontier Airlines Inc.  
 8 FL AirTran Airways Corporation  
 9 HA Hawaiian Airlines Inc.  
10 MQ Envoy Air  
11 OO SkyWest Airlines Inc.  
12 UA United Air Lines Inc.  
13 US US Airways Inc.  
14 VX Virgin America  
15 WN Southwest Airlines Co.  
16 YV Mesa Airlines Inc.

### 데이터의 관계

knitr::include\_graphics("http://r4ds.had.co.nz/diagrams/relational-nycflights.png")



## 키(keys)

한 쌍의 데이터 테이블을 연결해주는 변수(들)을 키라고 한다. 키는 관찰치를 고유하게 식별해주는 변수로써 간단한 경우는 하나의 변수로 충분하다. 예를 들어 planes 데이터에서 각 비행기는 tailnum으로 식별된다. 반면 여러 변수들이 필요한 경우도 있다. 예를 들어 weather 데이터에서 관찰치를 식별하기 위해서는 year, month, day, hour 및 origin의 다섯 개의 변수가 필요하다.

### 키의 종류

* 기본키(primary key): 기본키는 해당 테이블에서 관찰치를 고유하게 식별해주는 변수이다. 예를 들어 planes$tailnum 은 planes 데이터에서 각 비행기를 고유하게 식별해주는 고유키이다.
* 외래키(foreign key): 다른 데이터 테이블의 관찰치를 고유하게 식별해주는 변수이다. 예를 들어 flights 테이블에서 filghts$tailnum 은 각 flight를 고유한 비행기와 매치시켜 주는 외래키이다.

어떤 변수는 기본키이자 외래키일 수도 있다. 예를 들어 origin은 weather데이터의 기본키 중 하나이자 airport 테이블과 연결해주는 외래키이다.

### 기본키의 확인

하나의 테이블에서 기본키를 식별한 경우, 정말로 그 기본키가 각 관찰치를 고우하게 식별하는지 확인해 볼 필요가 있다. 한가지 방법은 기본키를 count()한 후 n 값이 1이상 것을 찾아보면 된다.

예제

1. planes 데이터에서 tailnum을 count하여 n값이 1이상인 것을 filter해보자
2. weather데이터의 기본키를 확인해보자

planes

# A tibble: 3,322 x 9  
 tailnum year type manufacturer model  
 <chr> <int> <chr> <chr> <chr>  
 1 N10156 2004 Fixed wing multi engine EMBRAER EMB-145XR  
 2 N102UW 1998 Fixed wing multi engine AIRBUS INDUSTRIE A320-214  
 3 N103US 1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214  
 4 N104UW 1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214  
 5 N10575 2002 Fixed wing multi engine EMBRAER EMB-145LR  
 6 N105UW 1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214  
 7 N107US 1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214  
 8 N108UW 1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214  
 9 N109UW 1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214  
10 N110UW 1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214  
# ... with 3,312 more rows, and 4 more variables: engines <int>,  
# seats <int>, speed <int>, engine <chr>

planes %>%  
 count(tailnum) %>%  
 filter(n>1)

# A tibble: 0 x 2  
# ... with 2 variables: tailnum <chr>, n <int>

### 대체키(surrogate key)

어떤 경우는 테이블에 명백한 기본키가 없는 경우도 있다. 각각의 행이 관찰치이지만 변수들을 조합하여도 각 관찰치를 식별할 수 없는 경우이다. 예를 들어 flights 테이블에서 기본키는 어떤 것일까? 날짜(year, month, day) 와 flight 또는 날짜와 tailnum을 조합하여도 고유한 식별자는 되지 않는다.

flights %>%   
 count(year, month, day, flight) %>%   
 filter(n > 1)

# A tibble: 29,768 x 5  
 year month day flight n  
 <int> <int> <int> <int> <int>  
 1 2013 1 1 1 2  
 2 2013 1 1 3 2  
 3 2013 1 1 4 2  
 4 2013 1 1 11 3  
 5 2013 1 1 15 2  
 6 2013 1 1 21 2  
 7 2013 1 1 27 4  
 8 2013 1 1 31 2  
 9 2013 1 1 32 2  
10 2013 1 1 35 2  
# ... with 29,758 more rows

기본키가 없는 경우 mutate()와 row\_number()를 사용해서 기본키 역할을 하는 변수를 추가하는 것이 도움이 될 때가 있다. 이런 변수를 대체키(surrogate key)라고 하며 대체키는 데이터를 필터링한 후 원래의 데이터를 체크해볼때 유용하다.

### 관계(relation)

기본키와 다른 테이블의 외래키는 관계(relation)를 형성한다. 관계는 전형적으로 일대다 관계이다. 예를 들어 각 비행(flight)은 하나의 비행기(plane)을 갖지만 각 비행기는 여러 번의 비행을 한다. 어떤 데이터에서는 1대1의 관계를 보기도 하는데 이는 1대다 관계의 특수한 경우로 생각할 수 있다. 또한 다대일 및 일대다 관계로 부터 다대다 관계를 맺기도 한다. 예를 들어 이 데이터에서 airlines와 airports는 다대다 관계이다. rkr airline은 여러 공항으로 비행하고 각 공항에는 많은 항공사가 있다.

### 연습문제

1. flights 데이터에서 대체키를 추가하라.

flights

# A tibble: 336,776 x 19  
 year month day dep\_time sched\_dep\_time dep\_delay arr\_time  
 <int> <int> <int> <int> <int> <dbl> <int>  
 1 2013 1 1 517 515 2 830  
 2 2013 1 1 533 529 4 850  
 3 2013 1 1 542 540 2 923  
 4 2013 1 1 544 545 -1 1004  
 5 2013 1 1 554 600 -6 812  
 6 2013 1 1 554 558 -4 740  
 7 2013 1 1 555 600 -5 913  
 8 2013 1 1 557 600 -3 709  
 9 2013 1 1 557 600 -3 838  
10 2013 1 1 558 600 -2 753  
# ... with 336,766 more rows, and 12 more variables: sched\_arr\_time <int>,  
# arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,  
# origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,  
# minute <dbl>, time\_hour <dttm>

flights %>% mutate(no=row\_number())

# A tibble: 336,776 x 20  
 year month day dep\_time sched\_dep\_time dep\_delay arr\_time  
 <int> <int> <int> <int> <int> <dbl> <int>  
 1 2013 1 1 517 515 2 830  
 2 2013 1 1 533 529 4 850  
 3 2013 1 1 542 540 2 923  
 4 2013 1 1 544 545 -1 1004  
 5 2013 1 1 554 600 -6 812  
 6 2013 1 1 554 558 -4 740  
 7 2013 1 1 555 600 -5 913  
 8 2013 1 1 557 600 -3 709  
 9 2013 1 1 557 600 -3 838  
10 2013 1 1 558 600 -2 753  
# ... with 336,766 more rows, and 13 more variables: sched\_arr\_time <int>,  
# arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,  
# origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,  
# minute <dbl>, time\_hour <dttm>, no <int>

1. 다음 데이터에서 기본키를 확인하라

1.Lahman::Batting

2.babynames::babynames

3.nasaweather::atmos

4.fueleconomy::vehicles

5.ggplot2::diamonds

(경우에 따라 패키지를 설치하거나 도움말을 참조하라)

Lahman::Batting

## 변수 추가(Mutating joins)

### Mutating joins

Mutating join으로 두 테이블의 변수를 병합할 수 있다. 먼저 키를 매치 시킨후 변수를 다른 테이블로 복사한다. mutate()와 마찬가지로 변수를 오른쪽 끝에 추가하기 때문에 변수가 많은 경우 추가된 변수를 잘 볼 수 없다. mutating join의 동작을 잘 보기 위해 flights 데이터의 변수를 줄여 flights2에 저장한다.

flights2 <- flights %>%  
 select(year:day, hour, origin, dest, tailnum, carrier)  
flights2

# A tibble: 336,776 x 8  
 year month day hour origin dest tailnum carrier  
 <int> <int> <int> <dbl> <chr> <chr> <chr> <chr>  
 1 2013 1 1 5 EWR IAH N14228 UA  
 2 2013 1 1 5 LGA IAH N24211 UA  
 3 2013 1 1 5 JFK MIA N619AA AA  
 4 2013 1 1 5 JFK BQN N804JB B6  
 5 2013 1 1 6 LGA ATL N668DN DL  
 6 2013 1 1 5 EWR ORD N39463 UA  
 7 2013 1 1 6 EWR FLL N516JB B6  
 8 2013 1 1 6 LGA IAD N829AS EV  
 9 2013 1 1 6 JFK MCO N593JB B6  
10 2013 1 1 6 LGA ORD N3ALAA AA  
# ... with 336,766 more rows

flights2에 항공사의이름을 추가해보자 flights2데이터에 airlines 데이터를 left join()으로 병합한다.

flights2 %>%  
 select(-origin, -dest) %>%  
 left\_join(airlines, by ="carrier")

# A tibble: 336,776 x 7  
 year month day hour tailnum carrier name  
 <int> <int> <int> <dbl> <chr> <chr> <chr>  
 1 2013 1 1 5 N14228 UA United Air Lines Inc.  
 2 2013 1 1 5 N24211 UA United Air Lines Inc.  
 3 2013 1 1 5 N619AA AA American Airlines Inc.  
 4 2013 1 1 5 N804JB B6 JetBlue Airways  
 5 2013 1 1 6 N668DN DL Delta Air Lines Inc.  
 6 2013 1 1 5 N39463 UA United Air Lines Inc.  
 7 2013 1 1 6 N516JB B6 JetBlue Airways  
 8 2013 1 1 6 N829AS EV ExpressJet Airlines Inc.  
 9 2013 1 1 6 N593JB B6 JetBlue Airways  
10 2013 1 1 6 N3ALAA AA American Airlines Inc.  
# ... with 336,766 more rows

airlines데이터를 flights2 데이터에 join시키면 name 변수가 추가된다.

left\_join()함수를 쓰지 않고 base R의 subset과 mutate()를 써서 같은 작업을 할 수 있다.

flights2 %>%  
 select(-origin, -dest) %>%   
 mutate(name = airlines$name[match(carrier, airlines$carrier)])

# A tibble: 336,776 x 7  
 year month day hour tailnum carrier name  
 <int> <int> <int> <dbl> <chr> <chr> <chr>  
 1 2013 1 1 5 N14228 UA United Air Lines Inc.  
 2 2013 1 1 5 N24211 UA United Air Lines Inc.  
 3 2013 1 1 5 N619AA AA American Airlines Inc.  
 4 2013 1 1 5 N804JB B6 JetBlue Airways  
 5 2013 1 1 6 N668DN DL Delta Air Lines Inc.  
 6 2013 1 1 5 N39463 UA United Air Lines Inc.  
 7 2013 1 1 6 N516JB B6 JetBlue Airways  
 8 2013 1 1 6 N829AS EV ExpressJet Airlines Inc.  
 9 2013 1 1 6 N593JB B6 JetBlue Airways  
10 2013 1 1 6 N3ALAA AA American Airlines Inc.  
# ... with 336,766 more rows

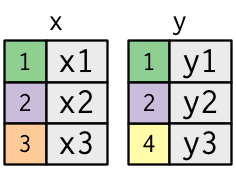
하지만 이 방법으로 여러 변수들을 match시키기는 어렵다. dplyr의 left\_join() 함수가 보다 직관적이며 속도가 빠르고 범용성이 있다.

### join의 이해

다음과 같이 x, y 두개의 테이블이 있다고 하자.

x <- tribble(  
 ~key, ~val\_x,  
 1, "x1",  
 2, "x2",  
 3, "x3"  
)  
y <- tribble(  
 ~key, ~val\_y,  
 1, "y1",  
 2, "y2",  
 4, "y3"  
)

knitr::include\_graphics("http://r4ds.had.co.nz/diagrams/join-setup.png")



색깔이 있는 열이 key 변수이다.

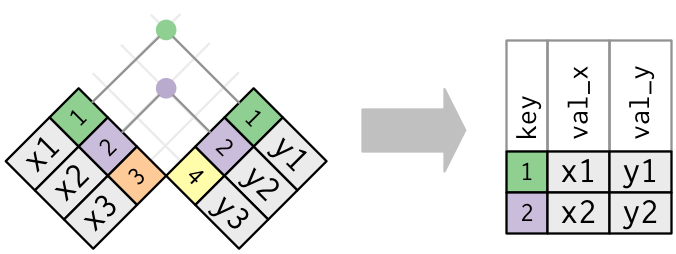
### Inner join

Inner join은 가장 간단한 join으로 두 테이블의 키가 일치하는 관찰치만 모아준다.

inner\_join(x,y,by="key")

# A tibble: 2 x 3  
 key val\_x val\_y  
 <dbl> <chr> <chr>  
1 1 x1 y1  
2 2 x2 y2

knitr::include\_graphics("http://r4ds.had.co.nz/diagrams/join-inner.png")



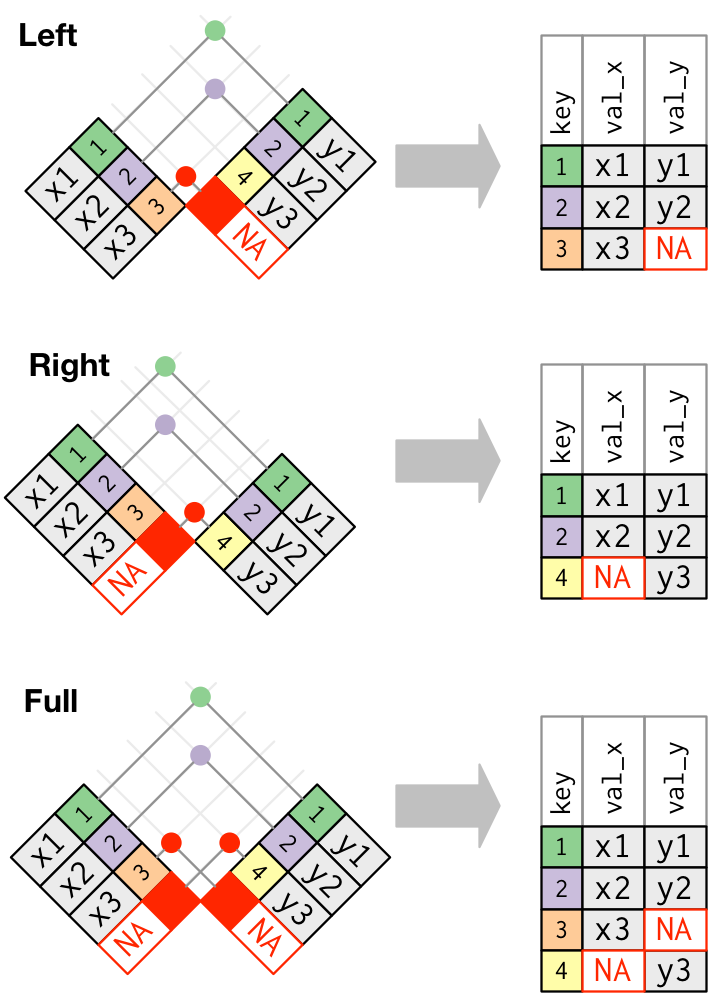
inner join의 가장 큰 특징은 매칭되지 않는 행은 포함되지 않는다는 것이다. 일반적으로 inner join은 관찰치를 쉽게 잃어버리기 때문에 보통 분석에는 적합하지 않다.

### Outer join

Inner join은 양 테이블에 모두 존재하는 관찰치만 유지시킨다. Outer join은 최소한 한 테이블의 관찰치를 보존한다. Outer join에는 세가지 type 이 있다.

* Left join은 x 의 모든 관찰치를 보존한다.
* Right join은 y 의 모든 관찰치를 보존한다.
* Full join은 x와 y 의 모든 관찰치를 보존한다.

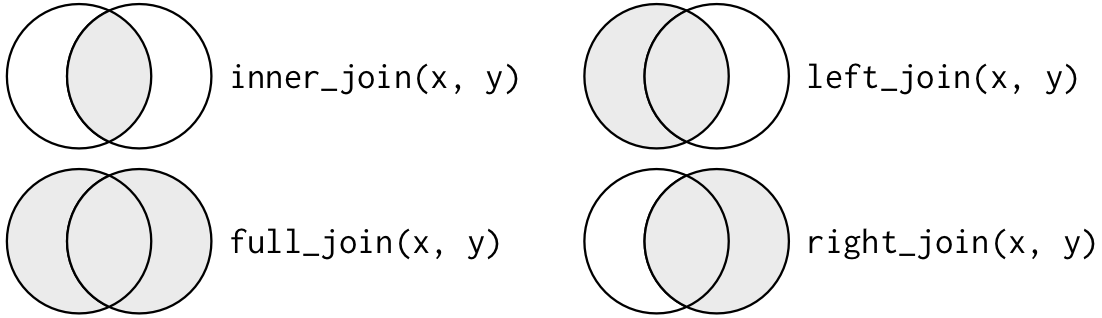
knitr::include\_graphics("http://r4ds.had.co.nz/diagrams/join-outer.png")



이중 가장 많이 사용되는 것은 left join이다. left join은 매칭되는 데이터가 없어도 original data를 보존하므로 다른 테이블에서 추가적인 데이터를 찾아볼때 유용하다. 따라서 디폴트로 left join을 사용하면서 다른 방법을 사용할 특별한 이유가 있을떄에만 다른 방법을 사용하자.

이를 벤 다이아그램으로 표시해보면 다음과 같다.

knitr::include\_graphics("http://r4ds.had.co.nz/diagrams/join-venn.png")

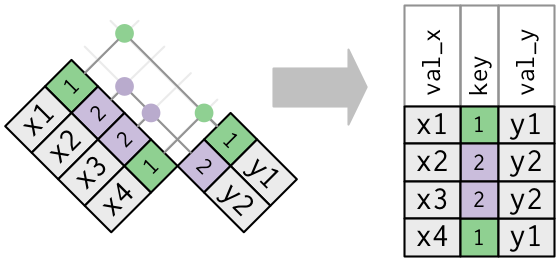


### 중복되는 키(Duplicate keys)

지금까지 살펴본 것은 키가 중복되지 않는 것을 가정했다. 하지만 키가 중복되는 경우도 있는데 두 가지로 나눌 수 있다.

1. 한 테이블에 중복되는 키가 있는 경우.

knitr::include\_graphics("http://r4ds.had.co.nz/diagrams/join-one-to-many.png")

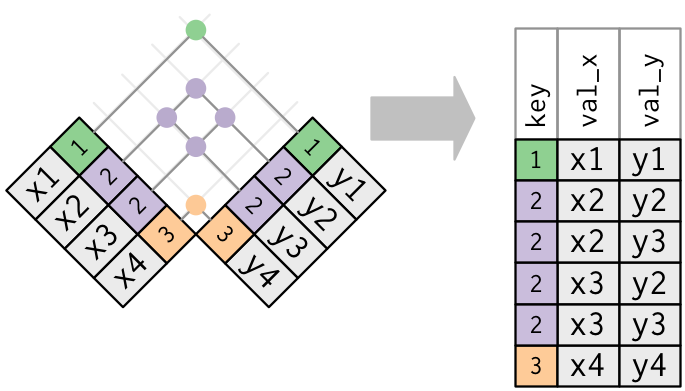


x <- tribble(  
 ~key, ~val\_x,  
 1, "x1",  
 2, "x2",  
 2, "x3",  
 1, "x4"  
)  
y <- tribble(  
 ~key, ~val\_y,  
 1, "y1",  
 2, "y2"  
)  
left\_join(x, y, by = "key")

# A tibble: 4 x 3  
 key val\_x val\_y  
 <dbl> <chr> <chr>  
1 1 x1 y1  
2 2 x2 y2  
3 2 x3 y2  
4 1 x4 y1

1. 두 테이블에 모두 중복되는 키가 있는 경우. 이런 경우는 양쪽 테이블에 모두 관찰치를 고우하게 식별하는 키가 없으므로 보통 에러인 경우가 많다. 이 경우는 모는 가능한 조합이 만들어진다.

knitr::include\_graphics("http://r4ds.had.co.nz/diagrams/join-many-to-many.png")



x <- tribble(  
 ~key, ~val\_x,  
 1, "x1",  
 2, "x2",  
 2, "x3",  
 3, "x4"  
)  
y <- tribble(  
 ~key, ~val\_y,  
 1, "y1",  
 2, "y2",  
 2, "y3",  
 3, "y4"  
)  
left\_join(x, y, by = "key")

# A tibble: 6 x 3  
 key val\_x val\_y  
 <dbl> <chr> <chr>  
1 1 x1 y1  
2 2 x2 y2  
3 2 x2 y3  
4 2 x3 y2  
5 2 x3 y3  
6 3 x4 y4

### 키 열(들)을 정의하기

지금까지는 테이블들은 하나의 변수로 연결되었고 그 변수는 양 테이블에 같은 이름으로 존재했다. 이러한 경우 by = "key"형식으로 코딩할 수 있다. 다른 방법으로 테이블들을 연결할때는 by인수에 다른 값을 사용할 수 있다.

* 디폴트 값인 by = NULL은 양 테이블에 모두 나타나는 모든 변수들을 사용하는데 이 경우 자연조인(natural join)이라고도 한다. 예를 들어 flights와 weather 테이블은 공통된 변수들로 year, month, day, hour, origin을 가지고 있다.

flights2 %>%  
 left\_join(weather)

# A tibble: 336,776 x 18  
 year month day hour origin dest tailnum carrier temp dewp humid  
 <dbl> <dbl> <int> <dbl> <chr> <chr> <chr> <chr> <dbl> <dbl> <dbl>  
 1 2013 1 1 5 EWR IAH N14228 UA NA NA NA  
 2 2013 1 1 5 LGA IAH N24211 UA NA NA NA  
 3 2013 1 1 5 JFK MIA N619AA AA NA NA NA  
 4 2013 1 1 5 JFK BQN N804JB B6 NA NA NA  
 5 2013 1 1 6 LGA ATL N668DN DL 39.92 26.06 57.33  
 6 2013 1 1 5 EWR ORD N39463 UA NA NA NA  
 7 2013 1 1 6 EWR FLL N516JB B6 39.02 26.06 59.37  
 8 2013 1 1 6 LGA IAD N829AS EV 39.92 26.06 57.33  
 9 2013 1 1 6 JFK MCO N593JB B6 39.02 26.06 59.37  
10 2013 1 1 6 LGA ORD N3ALAA AA 39.92 26.06 57.33  
# ... with 336,766 more rows, and 7 more variables: wind\_dir <dbl>,  
# wind\_speed <dbl>, wind\_gust <dbl>, precip <dbl>, pressure <dbl>,  
# visib <dbl>, time\_hour <dttm>

* 문자형 벡터를 by = "x"와 같이 사용할 수 있다. 자연조인과 비슷하지만 공통뵌 변수의 일부만 사용한다. 예를 들어 flights와 planes는 공통적으로 year변수가 있으나 서로 다른 의미로 사용하므로 tailnum만으로 조인한다.

flights2 %>%  
 left\_join(planes,by="tailnum")

# A tibble: 336,776 x 16  
 year.x month day hour origin dest tailnum carrier year.y  
 <int> <int> <int> <dbl> <chr> <chr> <chr> <chr> <int>  
 1 2013 1 1 5 EWR IAH N14228 UA 1999  
 2 2013 1 1 5 LGA IAH N24211 UA 1998  
 3 2013 1 1 5 JFK MIA N619AA AA 1990  
 4 2013 1 1 5 JFK BQN N804JB B6 2012  
 5 2013 1 1 6 LGA ATL N668DN DL 1991  
 6 2013 1 1 5 EWR ORD N39463 UA 2012  
 7 2013 1 1 6 EWR FLL N516JB B6 2000  
 8 2013 1 1 6 LGA IAD N829AS EV 1998  
 9 2013 1 1 6 JFK MCO N593JB B6 2004  
10 2013 1 1 6 LGA ORD N3ALAA AA NA  
# ... with 336,766 more rows, and 7 more variables: type <chr>,  
# manufacturer <chr>, model <chr>, engines <int>, seats <int>,  
# speed <int>, engine <chr>

이때 year변수가 양쪽 테이블에 모두 있으므로 혼동을 피하기 위해 year.x, year.y로 접미사를 붙인다.

* by = c("a" = "b")와 같이 이름을 붙인 문자형 벡터를 사용할 수도 있다. 이 경우 테이블 x의 a 변수와 테이블 y의 b변수를 매칭한다. 결과물에는 x의 변수 이름을 사용한다.

예를 들어 flights 데이터와 위도 경도 좌표가 있는 airports 데이터를 병합하여 지도를 그릴 수 있다. 각 비행기록에는 출발공항(origin)과 도착공항(destination)이 있으므로 어느 공항을 원하는지 명확이 하여야 한다.

flights2 %>%  
 left\_join(airports,c("dest"="faa"))

# A tibble: 336,776 x 15  
 year month day hour origin dest tailnum carrier  
 <int> <int> <int> <dbl> <chr> <chr> <chr> <chr>  
 1 2013 1 1 5 EWR IAH N14228 UA  
 2 2013 1 1 5 LGA IAH N24211 UA  
 3 2013 1 1 5 JFK MIA N619AA AA  
 4 2013 1 1 5 JFK BQN N804JB B6  
 5 2013 1 1 6 LGA ATL N668DN DL  
 6 2013 1 1 5 EWR ORD N39463 UA  
 7 2013 1 1 6 EWR FLL N516JB B6  
 8 2013 1 1 6 LGA IAD N829AS EV  
 9 2013 1 1 6 JFK MCO N593JB B6  
10 2013 1 1 6 LGA ORD N3ALAA AA  
# ... with 336,766 more rows, and 7 more variables: name <chr>, lat <dbl>,  
# lon <dbl>, alt <int>, tz <dbl>, dst <chr>, tzone <chr>

flights2 %>%  
 left\_join(airports,c("origin"="faa"))

# A tibble: 336,776 x 15  
 year month day hour origin dest tailnum carrier  
 <int> <int> <int> <dbl> <chr> <chr> <chr> <chr>  
 1 2013 1 1 5 EWR IAH N14228 UA  
 2 2013 1 1 5 LGA IAH N24211 UA  
 3 2013 1 1 5 JFK MIA N619AA AA  
 4 2013 1 1 5 JFK BQN N804JB B6  
 5 2013 1 1 6 LGA ATL N668DN DL  
 6 2013 1 1 5 EWR ORD N39463 UA  
 7 2013 1 1 6 EWR FLL N516JB B6  
 8 2013 1 1 6 LGA IAD N829AS EV  
 9 2013 1 1 6 JFK MCO N593JB B6  
10 2013 1 1 6 LGA ORD N3ALAA AA  
# ... with 336,766 more rows, and 7 more variables: name <chr>, lat <dbl>,  
# lon <dbl>, alt <int>, tz <dbl>, dst <chr>, tzone <chr>

### 연습문제

1. 목적지에 따른 평균 지연시간을 계산하고 airports 데이터와 조인하여 지연의 공간적인 분포를 시각화하라. 미국 지도를 그리는 쉬운 방법은 다음과 같다.

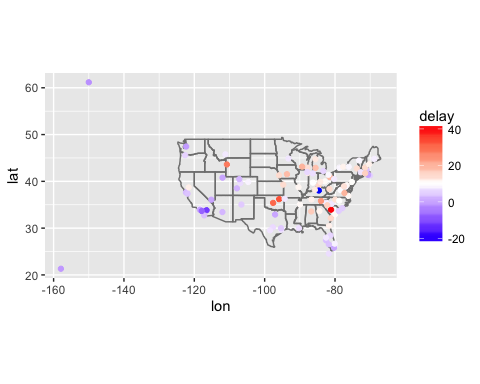
airports %>%  
 semi\_join(flights, c("faa"="dest")) %>%  
 ggplot(aes(lon, lat)) +  
 borders("state") +  
 geom\_point()+  
 coord\_quickmap()

 (semi\_join()함수는 다음 장에 나오므로 걱정할 필요가 없다.)

flights %>%  
 group\_by(dest) %>%  
 summarize(delay=mean(arr\_delay,na.rm=TRUE))

# A tibble: 105 x 2  
 dest delay  
 <chr> <dbl>  
 1 ABQ 4.381890  
 2 ACK 4.852273  
 3 ALB 14.397129  
 4 ANC -2.500000  
 5 ATL 11.300113  
 6 AUS 6.019909  
 7 AVL 8.003831  
 8 BDL 7.048544  
 9 BGR 8.027933  
10 BHM 16.877323  
# ... with 95 more rows

flights %>%  
 group\_by(dest) %>%  
 summarize(delay=mean(arr\_delay,na.rm=TRUE)) %>%  
 left\_join(airports, by=c("dest"="faa")) %>%  
 ggplot(aes(lon, lat,colour=delay)) +  
 borders("state") +  
 geom\_point() +  
 scale\_color\_gradientn(colours=c("blue","white","red"))+  
 coord\_quickmap()



1. flights2 데이터에 origin과 destination의 위치(lon과 lat)를 추가하라.

flights2

# A tibble: 336,776 x 8  
 year month day hour origin dest tailnum carrier  
 <int> <int> <int> <dbl> <chr> <chr> <chr> <chr>  
 1 2013 1 1 5 EWR IAH N14228 UA  
 2 2013 1 1 5 LGA IAH N24211 UA  
 3 2013 1 1 5 JFK MIA N619AA AA  
 4 2013 1 1 5 JFK BQN N804JB B6  
 5 2013 1 1 6 LGA ATL N668DN DL  
 6 2013 1 1 5 EWR ORD N39463 UA  
 7 2013 1 1 6 EWR FLL N516JB B6  
 8 2013 1 1 6 LGA IAD N829AS EV  
 9 2013 1 1 6 JFK MCO N593JB B6  
10 2013 1 1 6 LGA ORD N3ALAA AA  
# ... with 336,766 more rows

airports2 <- airports %>%  
 select(c(1,3,4))  
  
flights2 %>%  
 left\_join(airports2, by=c("origin"="faa")) %>%  
 left\_join(airports2, by=c("dest"="faa"))

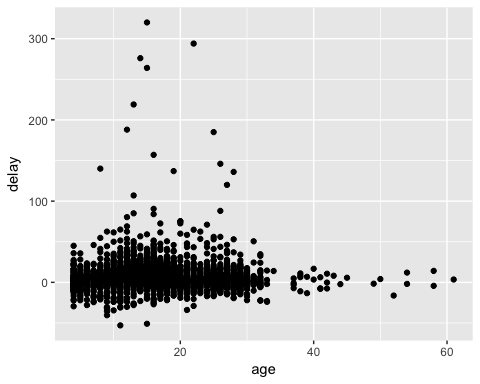
# A tibble: 336,776 x 12  
 year month day hour origin dest tailnum carrier lat.x lon.x  
 <int> <int> <int> <dbl> <chr> <chr> <chr> <chr> <dbl> <dbl>  
 1 2013 1 1 5 EWR IAH N14228 UA 40.69250 -74.16867  
 2 2013 1 1 5 LGA IAH N24211 UA 40.77725 -73.87261  
 3 2013 1 1 5 JFK MIA N619AA AA 40.63975 -73.77893  
 4 2013 1 1 5 JFK BQN N804JB B6 40.63975 -73.77893  
 5 2013 1 1 6 LGA ATL N668DN DL 40.77725 -73.87261  
 6 2013 1 1 5 EWR ORD N39463 UA 40.69250 -74.16867  
 7 2013 1 1 6 EWR FLL N516JB B6 40.69250 -74.16867  
 8 2013 1 1 6 LGA IAD N829AS EV 40.77725 -73.87261  
 9 2013 1 1 6 JFK MCO N593JB B6 40.63975 -73.77893  
10 2013 1 1 6 LGA ORD N3ALAA AA 40.77725 -73.87261  
# ... with 336,766 more rows, and 2 more variables: lat.y <dbl>,  
# lon.y <dbl>

1. 비행기의 나이와 연착시간은 관계가 있는가?

flights %>%   
 group\_by(tailnum) %>%  
 summarise(delay=mean(arr\_delay,na.rm=TRUE))

# A tibble: 4,044 x 2  
 tailnum delay  
 <chr> <dbl>  
 1 D942DN 31.5000000  
 2 N0EGMQ 9.9829545  
 3 N10156 12.7172414  
 4 N102UW 2.9375000  
 5 N103US -6.9347826  
 6 N104UW 1.8043478  
 7 N10575 20.6914498  
 8 N105UW -0.2666667  
 9 N107US -5.7317073  
10 N108UW -1.2500000  
# ... with 4,034 more rows

flights2 <- flights %>%   
 group\_by(tailnum) %>%  
 summarise(delay=mean(arr\_delay,na.rm=TRUE)) %>%  
 left\_join(planes) %>%  
 mutate(age=2017-year) %>%  
 select(1:3,"age")  
   
flights2 %>%  
 ggplot(aes(age,delay)) + geom\_point()

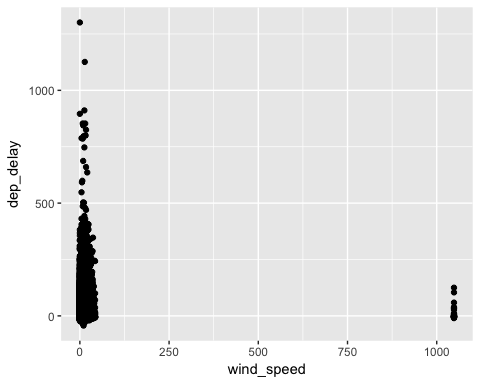


1. 어떤 기후조건이 보다 많은 지연을 일으키는가?

flights %>%  
 left\_join(weather)

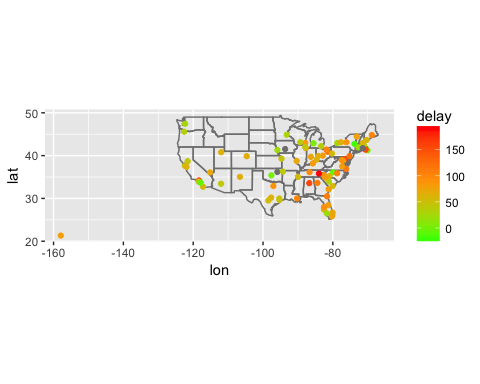
# A tibble: 336,776 x 28  
 year month day dep\_time sched\_dep\_time dep\_delay arr\_time  
 <dbl> <dbl> <int> <int> <int> <dbl> <int>  
 1 2013 1 1 517 515 2 830  
 2 2013 1 1 533 529 4 850  
 3 2013 1 1 542 540 2 923  
 4 2013 1 1 544 545 -1 1004  
 5 2013 1 1 554 600 -6 812  
 6 2013 1 1 554 558 -4 740  
 7 2013 1 1 555 600 -5 913  
 8 2013 1 1 557 600 -3 709  
 9 2013 1 1 557 600 -3 838  
10 2013 1 1 558 600 -2 753  
# ... with 336,766 more rows, and 21 more variables: sched\_arr\_time <int>,  
# arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,  
# origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,  
# minute <dbl>, time\_hour <dttm>, temp <dbl>, dewp <dbl>, humid <dbl>,  
# wind\_dir <dbl>, wind\_speed <dbl>, wind\_gust <dbl>, precip <dbl>,  
# pressure <dbl>, visib <dbl>

delay <- flights %>%  
 left\_join(weather) %>%  
 select("dep\_delay",20:28)  
  
delay %>% ggplot(aes(wind\_speed, dep\_delay)) +geom\_point()

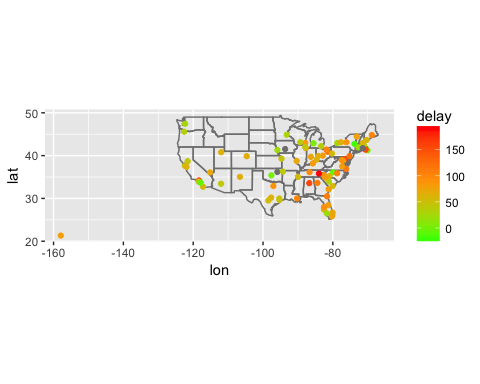


1. 2013년 6월 13일에 어떤 일이 일어났는가? 연착의 공간적 패턴을 표시하고 구글을 사용하여 날씨의 corss-referece를 찾아보라

flights %>%   
 filter(year==2013, month==6, day==13) %>%  
 group\_by(dest) %>%  
 summarize(delay=mean(arr\_delay,na.rm=TRUE)) %>%  
 left\_join(airports,by=c("dest"="faa")) %>%  
 ggplot(aes(lon, lat,colour=delay)) +  
 scale\_colour\_gradientn(colors=c("green","orange","red"))+  
 borders("state") +  
 geom\_point()+  
 coord\_quickmap()



flights %>%   
 filter(year==2013, month==6, day==13) %>%  
 group\_by(dest) %>%  
 summarize(delay=mean(arr\_delay,na.rm=TRUE)) %>%  
 left\_join(airports,by=c("dest"="faa")) %>%  
 ggplot(aes(lon, lat,colour=delay)) +  
 scale\_colour\_gradientn(colors=c("green","orange","red"))+  
 borders("state") +  
 geom\_point()+  
 coord\_quickmap()



### base R 과의 비교

base::merge()함수는 mutating join을 모두 수행할 수 있다.

|  |  |
| --- | --- |
| dplyr | merge |
| inner\_join(x,y) | merge(x,y) |
| left\_join(x,y) | merge(x,y,all.x=TRUE) |
| right\_join(x,y) | merge(x,y,all.y=TRUE) |
| full\_join(x,y) | merge(x,y,all.x=TRUE,all.y=TRUE) |

dplyr의 장점은 보다 직관적으로 의미를 전달한다는 점이다. 또한 dplyr 의 join이 속도가 빠르며 행의 순서를 흩뜨리지 않는다.

### SQL 과의 비교

dplyr 함수는 SQL에서 영감을 받은 것으로 거의 비슷하다.

|  |  |
| --- | --- |
| dplyr | SQL |
| inner\_join(x,y, by= "z") | SELECT \* FROM x INNER JOIN y USING (z) |
| left\_join(x,y, by= "z") | SELECT \* FROM x LEFT OUTER JOIN y USING (z) |
| right\_join(x,y, by= "z") | SELECT \* FROM x RIGHT OUTER JOIN y USING (z) |
| full\_join(x,y, by= "z") | SELECT \* FROM x FULL OUTER JOIN y USING (z) |

SQL 문에서 INNER 와 OUTER는 생략 가능하다. 두 테이블의 변수 이름이 다를때는 문법이 약간 다른데 inner\_join(x,y,by=c("a"="b")) 는 SQL에서는 SELECT \* FROM x INNER JOIN y ON x.a = y.b로 사용한다. SQL은 dplyr과는 달리 equalty 가 아닌 join type(non-equijoins)도 지원한다.

## 필터링 조인(Filtering joins)

필터링 조인은 mutating joins와 같은 방법으로 관찰치를 매치하지만 변수가 아니라 관찰치에 영향을 미친다. 여기에는 두가지 타입이 있다.

* semi\_join(x,y)는 y에 매치되는 x의 모든 관찰치 유지
* anti\_join(x,y)는 y에 매치되는 x의 모든 관찰치 제거

Semi-join은 필터된 요약테이블을 원래 행에 매치시킬때 유용하다. 예를 들어 가장 많이 찾는 상위 10개 목적지를 찾는다고 하면 다음과 같이 할 수 있다.

top\_dest <- flights %>%  
 count(dest, sort=TRUE) %>%  
 head(10)  
  
top\_dest

# A tibble: 10 x 2  
 dest n  
 <chr> <int>  
 1 ORD 17283  
 2 ATL 17215  
 3 LAX 16174  
 4 BOS 15508  
 5 MCO 14082  
 6 CLT 14064  
 7 SFO 13331  
 8 FLL 12055  
 9 MIA 11728  
10 DCA 9705

이제 top\_dest로 비행한 flights를 찾으려고 하면 먼저 다음과 같이 filter()하는 것을 생각해볼 수 있다.

flights %>%  
 filter(dest %in% top\_dest$dest)

# A tibble: 141,145 x 19  
 year month day dep\_time sched\_dep\_time dep\_delay arr\_time  
 <int> <int> <int> <int> <int> <dbl> <int>  
 1 2013 1 1 542 540 2 923  
 2 2013 1 1 554 600 -6 812  
 3 2013 1 1 554 558 -4 740  
 4 2013 1 1 555 600 -5 913  
 5 2013 1 1 557 600 -3 838  
 6 2013 1 1 558 600 -2 753  
 7 2013 1 1 558 600 -2 924  
 8 2013 1 1 558 600 -2 923  
 9 2013 1 1 559 559 0 702  
10 2013 1 1 600 600 0 851  
# ... with 141,135 more rows, and 12 more variables: sched\_arr\_time <int>,  
# arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,  
# origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,  
# minute <dbl>, time\_hour <dttm>

하지만 이 방법은 여러 개의 변수에 확장해서 사용하기는 어렵다. 예를 들어 평균 연착이 가장 많은 10일을 찾았다고 해보자. 이때 이와 같은 filter방법으로 year, month, day를 flights 데이터에서 매칭하기는 어렵다.

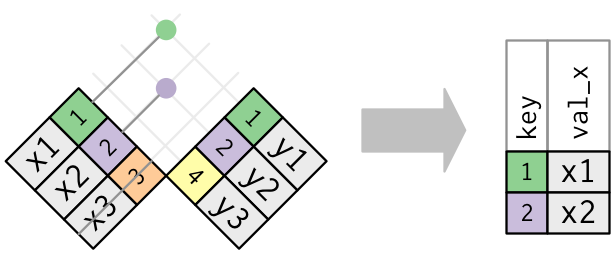
semi-join을 사용하면 두 테이블을 mutating join 같이 연결하지만 새 변수를 추가하는 것이 아니라 y에 매치가 되는 x의 관찰치만을 남긴다.

flights %>%  
 semi\_join(top\_dest)

# A tibble: 141,145 x 19  
 year month day dep\_time sched\_dep\_time dep\_delay arr\_time  
 <int> <int> <int> <int> <int> <dbl> <int>  
 1 2013 1 1 554 558 -4 740  
 2 2013 1 1 558 600 -2 753  
 3 2013 1 1 608 600 8 807  
 4 2013 1 1 629 630 -1 824  
 5 2013 1 1 656 700 -4 854  
 6 2013 1 1 709 700 9 852  
 7 2013 1 1 715 713 2 911  
 8 2013 1 1 739 745 -6 918  
 9 2013 1 1 749 710 39 939  
10 2013 1 1 828 830 -2 1027  
# ... with 141,135 more rows, and 12 more variables: sched\_arr\_time <int>,  
# arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,  
# origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,  
# minute <dbl>, time\_hour <dttm>

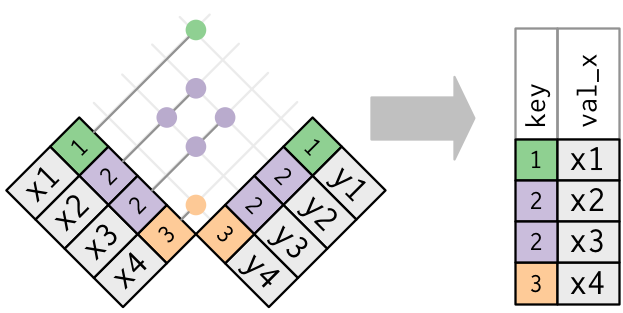
Semi-join을 그림으로 나타내면 다음과 같다.

knitr::include\_graphics("http://r4ds.had.co.nz/diagrams/join-semi.png")



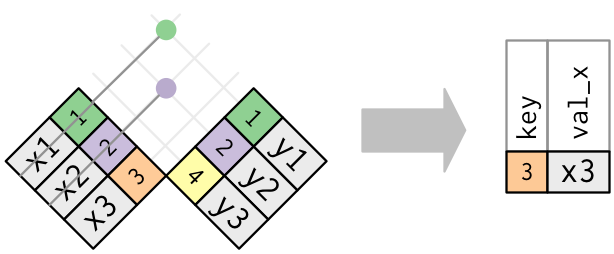
매치의 존재가 중요하고 중복되는 자료가 있어도 mutating join처럼 중복되지는 않는다.

knitr::include\_graphics("http://r4ds.had.co.nz/diagrams/join-semi-many.png")



Semi-join의 역은 anti-join 이다. Anti-join은 match가 되지 않는 행만을 유지한다.

knitr::include\_graphics("http://r4ds.had.co.nz/diagrams/join-anti.png")



Anti-join은 mismatch를 진단할 때 유용하다. 예를 들어 flights와 planes를 연결할 때 planes 데이터에 match 되지 않는 비행데이터가 얼마나 되는지 관심있다면 다음과 같이 할 수 있다.

flights %>%  
 anti\_join(planes,by="tailnum") %>%  
 count(tailnum, sort=TRUE)

# A tibble: 722 x 2  
 tailnum n  
 <chr> <int>  
 1 <NA> 2512  
 2 N725MQ 575  
 3 N722MQ 513  
 4 N723MQ 507  
 5 N713MQ 483  
 6 N735MQ 396  
 7 N0EGMQ 371  
 8 N534MQ 364  
 9 N542MQ 363  
10 N531MQ 349  
# ... with 712 more rows

### 연습문제

flights %>%  
 filter(is.na(tailnum))

# A tibble: 2,512 x 19  
 year month day dep\_time sched\_dep\_time dep\_delay arr\_time  
 <int> <int> <int> <int> <int> <dbl> <int>  
 1 2013 1 2 NA 1545 NA NA  
 2 2013 1 2 NA 1601 NA NA  
 3 2013 1 3 NA 857 NA NA  
 4 2013 1 3 NA 645 NA NA  
 5 2013 1 4 NA 845 NA NA  
 6 2013 1 4 NA 1830 NA NA  
 7 2013 1 5 NA 840 NA NA  
 8 2013 1 7 NA 820 NA NA  
 9 2013 1 8 NA 1645 NA NA  
10 2013 1 9 NA 755 NA NA  
# ... with 2,502 more rows, and 12 more variables: sched\_arr\_time <int>,  
# arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,  
# origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,  
# minute <dbl>, time\_hour <dttm>

1. flights 데이터 중에서 최소 100번 이상 비행한 비행기의 기록만 필터링하라.

common\_plane<-flights %>% count(tailnum,sort=TRUE) %>% filter(n>100)  
flights %>% semi\_join(common\_plane)

# A tibble: 229,202 x 19  
 year month day dep\_time sched\_dep\_time dep\_delay arr\_time  
 <int> <int> <int> <int> <int> <dbl> <int>  
 1 2013 1 2 NA 1545 NA NA  
 2 2013 1 2 NA 1601 NA NA  
 3 2013 1 3 NA 857 NA NA  
 4 2013 1 3 NA 645 NA NA  
 5 2013 1 4 NA 845 NA NA  
 6 2013 1 4 NA 1830 NA NA  
 7 2013 1 5 NA 840 NA NA  
 8 2013 1 7 NA 820 NA NA  
 9 2013 1 8 NA 1645 NA NA  
10 2013 1 9 NA 755 NA NA  
# ... with 229,192 more rows, and 12 more variables: sched\_arr\_time <int>,  
# arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,  
# origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,  
# minute <dbl>, time\_hour <dttm>

1. fueleconomy::vehicles 와 fueleconomy::common 데이터를 병합하여 가장 흔한 모델들의 기록을 찾아라.

fueleconomy::vehicles %>%  
 semi\_join(fueleconomy::common)

# A tibble: 14,531 x 12  
 id make model year class trans  
 <int> <chr> <chr> <int> <chr> <chr>  
 1 1833 Acura Integra 1986 Subcompact Cars Automatic 4-spd  
 2 1834 Acura Integra 1986 Subcompact Cars Manual 5-spd  
 3 3037 Acura Integra 1987 Subcompact Cars Automatic 4-spd  
 4 3038 Acura Integra 1987 Subcompact Cars Manual 5-spd  
 5 4183 Acura Integra 1988 Subcompact Cars Automatic 4-spd  
 6 4184 Acura Integra 1988 Subcompact Cars Manual 5-spd  
 7 5303 Acura Integra 1989 Subcompact Cars Automatic 4-spd  
 8 5304 Acura Integra 1989 Subcompact Cars Manual 5-spd  
 9 6442 Acura Integra 1990 Subcompact Cars Automatic 4-spd  
10 6443 Acura Integra 1990 Subcompact Cars Manual 5-spd  
# ... with 14,521 more rows, and 6 more variables: drive <chr>, cyl <int>,  
# displ <dbl>, fuel <chr>, hwy <int>, cty <int>

anti\_join(flights, airports, by = c("dest" = "faa"))

# A tibble: 7,602 x 19  
 year month day dep\_time sched\_dep\_time dep\_delay arr\_time  
 <int> <int> <int> <int> <int> <dbl> <int>  
 1 2013 1 1 2353 2359 -6 425  
 2 2013 1 2 2351 2359 -8 427  
 3 2013 1 3 2349 2359 -10 434  
 4 2013 1 4 2358 2359 -1 436  
 5 2013 1 5 14 2359 15 503  
 6 2013 1 6 2353 2359 -6 441  
 7 2013 1 7 49 2359 50 531  
 8 2013 1 8 2351 2359 -8 417  
 9 2013 1 9 2 2359 3 432  
10 2013 1 10 16 2359 17 447  
# ... with 7,592 more rows, and 12 more variables: sched\_arr\_time <int>,  
# arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,  
# origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,  
# minute <dbl>, time\_hour <dttm>

anti\_join(airports, flights, by = c("faa" = "dest"))

# A tibble: 1,357 x 8  
 faa name lat lon alt tz dst  
 <chr> <chr> <dbl> <dbl> <int> <dbl> <chr>  
 1 ZYP Penn Station 40.75050 -73.99350 35 -5 A  
 2 ZWU Washington Union Station 38.89746 -77.00643 76 -5 A  
 3 ZWI Wilmington Amtrak Station 39.73667 -75.55167 0 -5 A  
 4 ZVE New Haven Rail Station 41.29867 -72.92599 7 -5 A  
 5 ZUN Black Rock 35.08323 -108.79178 6454 -7 A  
 6 ZTY Boston Back Bay Station 42.34780 -71.07500 20 -5 A  
 7 ZTF Stamford Amtrak Station 41.04694 -73.54149 0 -5 A  
 8 ZSY Scottsdale Airport 33.62289 -111.91053 1519 -7 A  
 9 ZSF Springfield Amtrak Station 42.10600 -72.59305 65 -5 A  
10 ZRZ New Carrollton Rail Station 38.94800 -76.87190 39 -5 A  
# ... with 1,347 more rows, and 1 more variables: tzone <chr>

1. 각 비행기는 한 항공사에 의해 운행되므로 비행기와 항공사 간에 암묵적인 관계가 있을 것으로 생각할 수 있다. 이 가설을 확인하라.

flights %>%   
 select(carrier,tailnum) %>%  
 group\_by(tailnum) %>%  
 count(carrier) %>%  
 count(tailnum) %>%  
 filter(nn>1) -> dup  
  
flights %>%   
 select(carrier,tailnum) %>%  
 group\_by(tailnum) %>%  
 count(carrier) %>%  
 semi\_join(dup)

# A tibble: 41 x 3  
# Groups: tailnum [?]  
 tailnum carrier n  
 <chr> <chr> <int>  
 1 N146PQ 9E 8  
 2 N146PQ EV 36  
 3 N153PQ 9E 5  
 4 N153PQ EV 26  
 5 N176PQ 9E 7  
 6 N176PQ EV 21  
 7 N181PQ 9E 4  
 8 N181PQ EV 35  
 9 N197PQ 9E 2  
10 N197PQ EV 31  
# ... with 31 more rows