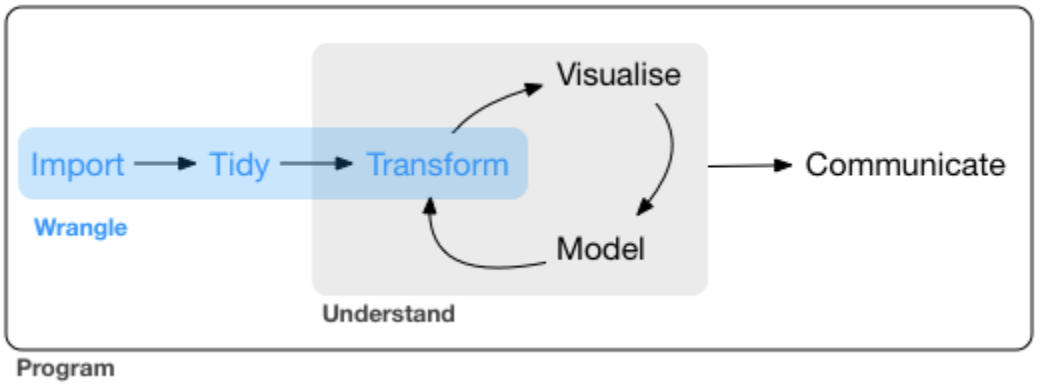
데이터 변형

문 건 웅

2017년 8월 26일

데이터 변형


Visualise
Import
Tidy
Transform
Communicate
Wrangle
Model
Understand
Program

Prerequisites

```
#install.packages("tidyverse")

library(tidyverse)
library(nycflights13)
```

Tibbles

Tibble은 데이터프레임을 개선시킨 데이터구조이다.

```
iris
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1            5.1         3.5          1.4         0.2     setosa
2            4.9         3.0          1.4         0.2     setosa
3            4.7         3.2          1.3         0.2     setosa
4            4.6         3.1          1.5         0.2     setosa
5            5.0         3.6          1.4         0.2     setosa
6            5.4         3.9          1.7         0.4     setosa
7            4.6         3.4          1.4         0.3     setosa
8            5.0         3.4          1.5         0.2     setosa
9            4.4         2.9          1.4         0.2     setosa
10           4.9         3.1          1.5         0.1     setosa
11           5.4         3.7          1.5         0.2     setosa
12           4.8         3.4          1.6         0.2     setosa
13           4.8         3.0          1.4         0.1     setosa
14           4.3         3.0          1.1         0.1     setosa
15           5.8         4.0          1.2         0.2     setosa
16           5.7         4.4          1.5         0.4     setosa
17           5.4         3.9          1.3         0.4     setosa
```

```
iris1=as_tibble(iris)
iris1
```

```
# A tibble: 150 x 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl>  <fctr>
 1          5.1         3.5          1.4         0.2  setosa
 2          4.9         3.0          1.4         0.2  setosa
 3          4.7         3.2          1.3         0.2  setosa
 4          4.6         3.1          1.5         0.2  setosa
 5          5.0         3.6          1.4         0.2  setosa
 6          5.4         3.9          1.7         0.4  setosa
 7          4.6         3.4          1.4         0.3  setosa
 8          5.0         3.4          1.5         0.2  setosa
 9          4.4         2.9          1.4         0.2  setosa
10          4.9         3.1          1.5         0.1  setosa
# ... with 140 more rows
```

데이터 변형

- 예제소개
- 조건에 맞는 관찰치 찾기 `filter()`
- 정렬하기 `arrange()`
- 원하는 변수만 고르기 `select()`
- 새로운 변수를 추가 `mutate()`
- 그룹별로 요약하기 `summarise()`
- 그룹별 변수추가(및 데이터 필터링)

데이터 소개

- *flights* : 2013년에 뉴욕시에서 출발한 모든 비행기록(336,776 flights)

```
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>
```

출력하기

```
print(flights,n=10,width=Inf)
```

```
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_tim
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int
 1  2013     1     1      517            515         2      830            81
 2  2013     1     1      533            529         4      850            83
 3  2013     1     1      542            540         2      923            85
 4  2013     1     1      544            545        -1     1004           102
 5  2013     1     1      554            600        -6      812            83
 6  2013     1     1      554            558        -4      740            72
 7  2013     1     1      555            600        -5      913            85
 8  2013     1     1      557            600        -3      709            72
 9  2013     1     1      557            600        -3      838            84
10  2013     1     1      558            600        -2      753            74
# ... with 3.368e+05 more rows
```

변수의 종류

- int 정수
- dbl 실수
- chr 문자형
- dttm 날짜-시간(날짜+시간)
- lgl 논리형 - TRUE or FALSE
- fctr 팩터형. 고정된 값을 갖는 범주형 변수
- date 날짜

dplyr의 다섯가지의 키 함수

- 조건에 맞는 관찰치 찾기: `filter()`
- 정렬하기: `arrange()`
- 원하는 변수만 고르기: `select()`
- 새로운 변수를 추가하기: `mutate()`
- 하나의 변수로 요약하기: `summarise()`

dplyr의 다섯가지의 키 함수

- 조건에 맞는 관찰치 찾기: `filter()`
- 정렬하기: `arrange()`
- 원하는 변수만 고르기: `select()`
- 새로운 변수를 추가하기: `mutate()`
- 하나의 변수로 요약하기: `summarise()`

One more thing

- 모든 함수를 그룹별로 적용하기 `group_by()`

사용하기

모든 함수의 사용법은 유사하다:

1. 첫번째 인수는 데이터프레임이다.
2. 두번째 이후의 인수는 그 데이터프레임으로 무엇을 할 것인지 기술한다.(변수이름 사용)
3. 결과는 새로운 데이터프레임이다.

외워봅시다: dplyr의 키 함수

- 조건에 맞는 관찰치 찾기: `filter()`
- 정렬하기: `arrange()`
- 원하는 변수만 고르기: `select()`
- 새로운 변수를 추가하기: `mutate()`
- 하나의 변수로 요약하기: `summarise()`
- 그룹별로 함수 적용: `group_by()`

Filter rows with filter()

filter() 함수는 관찰치들의 값을 기준으로 부분집합을 만들어준다. 함수의 첫번째 인수는 데이터프레임의 이름이고 두번째 이후의 인수들은 데이터를 골라내는 기준이다.

flights 데이터 중 1월 1일 의 데이타만 골라내려면:

```
filter(flights, month==1, day==1)
```

```
# A tibble: 842 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 832 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>,
```

filter() 함수는 filter를 적용하여 새로운 데이터프레임을 만들지만 원본을 바꾸지는 않는다. 따라서 filter를 적용한 새로운 데이터를 jan1이라는 이름으로 저장하려면 할당연산자 <- 를 사용하여 저장한다.

tip: <-는 control + - 또는 command + -로 입력할 수 있다.

```
jan1 <- filter(flights, month==1, day==1)
```

이때는 새로운 이름으로 할당이 되지만 인쇄가 되지는 않는다.

할당과 동시에 인쇄를 하려면 할당문 앞뒤를 괄호로 감싸주면 된다. 예를 들어 12월 25일의 비행기록을 dec25에 할당하면서 동시에 저장하려면 다음과 같이 한다.

```
(dec25 <- filter(flights, month==12, day==25))
```

```
# A tibble: 719 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013    12    25      456            500        -4      649
 2  2013    12    25      524            515         9      805
 3  2013    12    25      542            540         2      832
 4  2013    12    25      546            550        -4     1022
 5  2013    12    25      556            600        -4      730
 6  2013    12    25      557            600        -3      743
 7  2013    12    25      557            600        -3      818
 8  2013    12    25      559            600        -1      855
 9  2013    12    25      559            600        -1      849
10  2013    12    25      600            600         0      850
# ... with 709 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>
```

비교하기

R에서 사용할 수 있는 비교 연산자는 다음과 같다. `>`, `>=`, `<`, `<=`, `==(equal)`, `!=(not equal)`.

처음 R을 배우기 시작할 때 가장 범하기 쉬운 에러는 ==(equal)대신 =(할당연산자)를 사용하는 것이다. 예를 들어 1월의 비행기록을 고르려고 할때 =을 사용하게 되면 다음과 같은 에러가 난다.

```
filter(flights, month = 1)
```

```
Error: `month` (`month = 1`) must not be named, do you need `==`?
```

또 하나 비교 연산자 ==를 사용할 때 문제는 부동소수점(floating point) 문제이다.

```
sqrt(2)^2 == 2
```

```
[1] FALSE
```

```
1/49*49 == 1
```

```
[1] FALSE
```

또 하나 비교 연산자 ==를 사용할 때 문제는 부동소수점(floating point) 문제이다.

```
sqrt(2)^2 == 2
```

```
[1] FALSE
```

```
1/49*49 == 1
```

```
[1] FALSE
```

컴퓨터에서 사용하는 숫자는 부동소수점 방식으로 고정 소수점 방식보다 넓은 범위의 수를 나타낼 수 있어 과학기술 계산에 많이 이용되지만, 근삿값으로 표현된다. 따라서 실수끼리 비교할때 == 연산자 대신 near ()함수를 쓰면 된다.

```
 near(sqrt(2)^2,2)
```

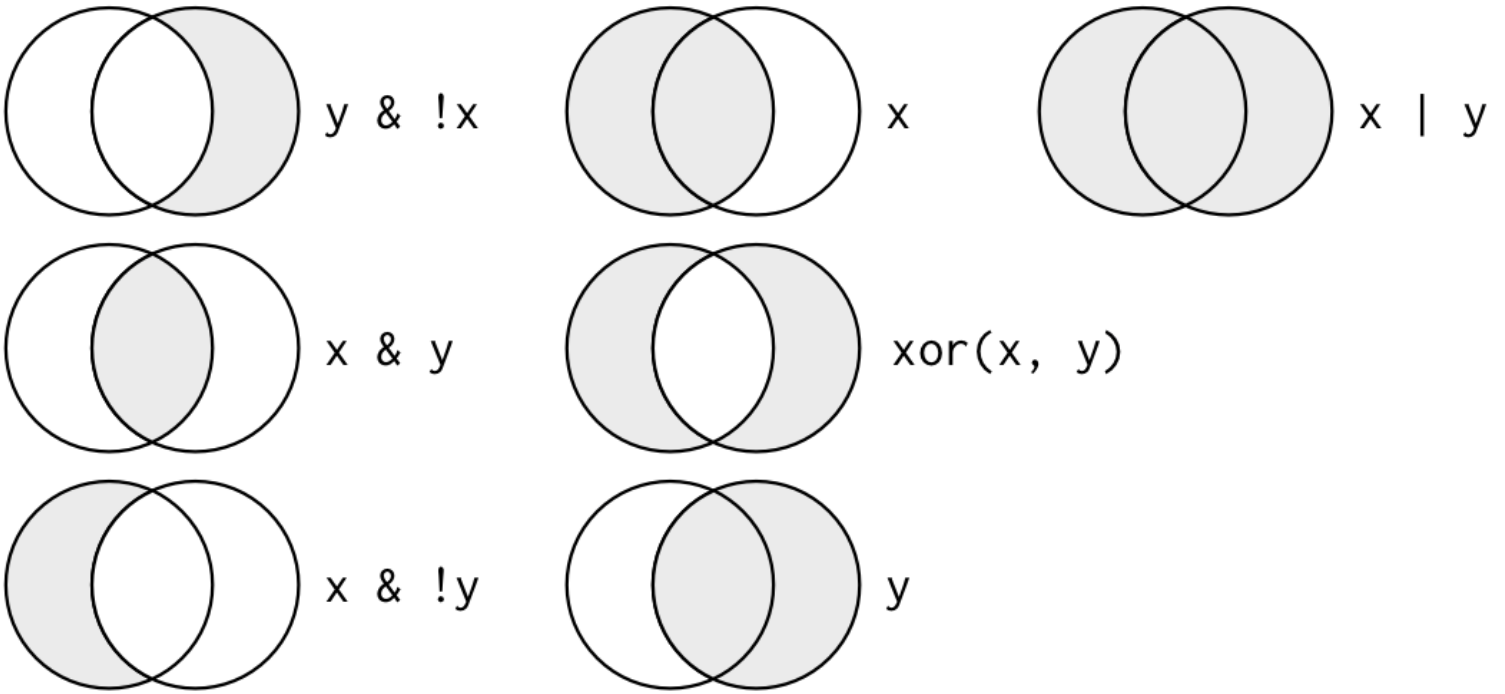
```
[1] TRUE
```

```
 near(1/49*49,1)
```

```
[1] TRUE
```

논리 연산자

filter()함수에 비교문을 여러 개 넣으면 and로 작동한다. 그 외의 논리 연산자로 &는 and, |는 or, 그리고 !는 not으로 작동한다. 그 외의 불린 연산자는 다음 그림을 참조한다.



flights 데이터에서 1월과 12월의 비행기록을 찾으려면 다음과 같이 한다.

```
filter(flights, month == 1 | month == 12)
```

```
# A tibble: 55,139 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 55,129 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 다음과 같이 쓸 수는 없다.

```
filter(flights, month == 1 | 12)
```

이때는 1|12 가 TRUE가 되므로 모든 자료가 된다. 이러한 방법으로 쓸 수 있는 연산자는 **포함(include) 연산자**이며 x %in% y와 같이 사용한다. 이 연산자는 x가 y의 값중 하나일때 참이 된다. 따라서 1월, 7월, 12월의 비행기록을 찾으려면 다음과 같이 할 수 있다.

```
filter(flights, month %in% c(1,7,12))
```

```
# A tibble: 84,564 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 84,554 more rows, and 12 more variables: sched_arr_time <int>,
#   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
```

드 모르간(De Morgan)의 법칙

- !(x & y) 는 !x | !y 와 같다.
- !(x | y) 는 !x & !y 와 같다.

(예) 출발 및 도착 지연이 2시간 이내인 모든 비행기록을 찾는다면?

```
filter(flights, !(arr_delay > 120 | dep_delay > 120))
filter(flights, arr_delay <= 120, dep_delay <= 120)
```

누락된 값(NA)

R의 비교문에서 가장 혼동하기 쉬운 것은 누락치(NA, not available)이다. NA는 전염력이 있다. NA 를 연산하는 경우 모두 NA 가 된다.

```
NA > 5
```

```
[1] NA
```

```
10 == NA
```

```
[1] NA
```

```
NA + 10
```

```
[1] NA
```

```
NA / 2
```

```
[1] NA
```

가장 혼동하기 쉬운 것은 다음 비교이다.

```
NA == NA
```

```
[1] NA
```

이것을 이해하기 위해서는 다음 예를 보자.

```
# 철수의 나이를 x라고 하고 나이를 모른다고 하자
x <- NA

# 영희의 나이를 y라고 하고 나이를 모른다고 하자
y <- NA

#철수와 영희가 나이가 같은가?
x == y
```

```
[1] NA
```

```
# 모른다!!
```

가장 혼동하기 쉬운 것은 다음 비교이다.

```
NA == NA
```

```
[1] NA
```

이것을 이해하기 위해서는 다음 예를 보자.

```
# 철수의 나이를 x라고 하고 나이를 모른다고 하자
x <- NA

# 영희의 나이를 y라고 하고 나이를 모른다고 하자
y <- NA

#철수와 영희가 나이가 같은가?
x == y
```

```
[1] NA
```

```
# 모른다!!
```

철수의 나이가 누락되어 있는가?

```
is.na(x)
```

```
[1] TRUE
```

filter()와 누락치

filter()는 주어진 조건이 참인 경우만 포함한다. 거짓이거나 누락치인 경우는 제외된다.

```
df <- tibble(x = c(1, NA, 3))
filter(df, x > 1)
```

```
# A tibble: 1 x 1
      x
  <dbl>
1     3
```

filter()와 누락치

filter()는 주어진 조건이 참인 경우만 포함한다. 거짓이거나 누락치인 경우는 제외된다.

```
df <- tibble(x = c(1, NA, 3))
filter(df, x > 1)
```

```
# A tibble: 1 x 1
      x
  <dbl>
1     3
```

누락치를 포함하려면?

```
filter(df, is.na(x) | x > 1)
```

```
# A tibble: 2 x 1
      x
  <dbl>
1    NA
2     3
```

Exercises

1.Find all flights that

- Had an arrival delay of two or more hours
- Flew to Houston(`IAH` or `HOU`)
- Were operated by United, American, or Delta
- Departed in summer (July, August, and September)
- Arrived more than two hours late, but didn't leave late
- Were delayed by at least an hour, but made up over 30 minutes in flight
- Departed between midnight and 6am (inclusive)

2.Another useful dplyr filtering helper is between(). What does it do? Can you use it to simplify the code needed to answer the previous challenges?

3.How many flights have a missing dep_time? What other variables are missing? What might these rows represent?

4.Why is `NA ^ 0` not missing? Why is `NA | TRUE` not missing? Why is `FALSE & NA` not missing? Can you figure out the general rule? (`NA * 0` is a tricky counterexample!)

Answers

1. Find all flights that

- Had an arrival delay of two or more hours

```
filter(flights,arr_delay >= 120)
```

- Flew to Houston(`IAH` or `HOU`)

```
filter(flights, dest %in% c("IAH","HOU"))
```

- Were operated by United, American, or Delta

```
filter(flights, carrier %in% c("UA","AA","DL"))
```

- Departed in summer (July, August, and September)

```
filter(flights, month >= 7, month <= 9)
```

- Arrived more than two hours late, but didn't leave late

```
filter(flights, arr_delay >= 120, dep_delay<=0 )
```

- Were delayed by at least an hour, but made up over 30 minutes in flight

```
filter(flights, arr_delay >= 60, dep_delay - arr_delay >= 30 )
```

- Departed between midnight and 6am (inclusive)

```
filter(flights, dep_time >= 0, dep_time <= 600 )
```

2.Another useful dplyr filtering helper is between(). What does it do? Can you use it to simplify the code needed to answer the previous challenges?

```
filter(flights, dep_time >=0, dep_time<=600)
filter(flights, between(dep_time ,0, 600))
```

3.How many flights have a missing dep_time? What other variables are missing? What might these rows represent?

```
filter(flights,is.na(dep_time))
```

데이터 정렬하기(Arrange rows) arrange()

arrange() 함수는 데이타프레임과 열 이름으로 데이타를 정렬한다.

```
arrange(flights, year, month, day)
```

```
# 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>
```

desc()을 사용하면 내림차순으로 정렬할 수 있다.

```
arrange(flights, desc(arr_delay))
```

```
# 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     9      641            900      1301     1242
 2  2013     6    15     1432           1935      1137     1607
 3  2013     1    10     1121           1635      1126     1239
 4  2013     9    20     1139           1845      1014     1457
 5  2013     7    22      845           1600      1005     1044
 6  2013     4    10     1100           1900       960     1342
 7  2013     3    17     2321            810       911      135
 8  2013     7    22     2257            759       898      121
 9  2013    12     5      756           1700       896     1058
10  2013     5     3     1133           2055       878     1250
# ... 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>
```

누락치(NA)는 가장 마지막에 정렬된다.

```
df <- tibble(x = c(5, 2, NA))
arrange(df, x)
```

```
# A tibble: 3 x 1
      x
  <dbl>
1     2
2     5
3    NA
```

```
arrange(df, desc(x))
```

```
# A tibble: 3 x 1
      x
  <dbl>
1     5
2     2
3    NA
```

Exercises

1.How could you use arrange() to sort all missing values to the start? (Hint: use is.na()).

2.Sort flights to find the most delayed flights. Find the flights that left earliest.

3.Sort flights to find the fastest flights.

4.Which flights travelled the longest? Which travelled the shortest?

Answers

1.How could you use arrange() to sort all missing values to the start? (Hint: use is.na()).

```
arrange(df, desc(x))
```

```
# A tibble: 3 x 1
      x
  <dbl>
1     5
2     2
3    NA
```

2.Sort flights to find the most delayed flights. Find the flights that left earliest.

```
arrange(flights, desc(arr_delay))
```

3.Sort flights to find the fastest flights.

```
arrange(flights,desc(distance/air_time))
top_n(flights,1,distance/air_time)
```

4.Which flights travelled the longest? Which travelled the shortest?

```
arrange(flights,desc(distance))
arrange(flights,desc(air_time))
top_n(flights,1,air_time)
```

원하는 열만 선택 select()

데이터의 변수가 수백개 또는 수천개인 경우가 종종 있다. 이런 경우 관심있는 변수들에 국한하여 작업을 진행하는 것이 좋다. select() 함수로 원하는 변수를 선택할 수 있다.

```
# 열 이름으로 선택하기
select(flights, year, month, day)
```

```
# A tibble: 336,776 x 3
    year month   day
   <int> <int> <int>
 1  2013     1     1
 2  2013     1     1
 3  2013     1     1
 4  2013     1     1
 5  2013     1     1
 6  2013     1     1
 7  2013     1     1
 8  2013     1     1
 9  2013     1     1
10  2013     1     1
# ... with 336,766 more rows
```

```r
# year부터 day 까지 열 선택하기
select(flights, year:day)
```

```
# A tibble: 336,776 x 3
    year month   day
   <int> <int> <int>
 1  2013     1     1
 2  2013     1     1
 3  2013     1     1
 4  2013     1     1
 5  2013     1     1
 6  2013     1     1
 7  2013     1     1
 8  2013     1     1
 9  2013     1     1
10  2013     1     1
# ... with 336,766 more rows
```

```
# year부터 day 까지 열을 제외하고 나머지 열 선택
select(flights, -(year:day))
```

```
# A tibble: 336,776 x 16
   dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
      <int>          <int>     <dbl>    <int>          <int>     <dbl>
 1      517            515         2      830            819        11
 2      533            529         4      850            830        20
 3      542            540         2      923            850        33
 4      544            545        -1     1004           1022       -18
 5      554            600        -6      812            837       -25
 6      554            558        -4      740            728        12
 7      555            600        -5      913            854        19
 8      557            600        -3      709            723       -14
 9      557            600        -3      838            846        -8
10      558            600        -2      753            745         8
# ... with 336,766 more rows, and 10 more variables: carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

select()와 함께 쓸 수 있는 도우미 함수들

- `starts_with("abc")`: “abc”로 시작하는 열이름 선택
- `ends_with("xyz")`: “xyz”으로 끝나는 열이름 선택
- `contains("ijk")`: “ijk”를 포함한 열이름 선택
- `matches()`: 정규표현식을 사용하여 변수이름 선택
- `num_range("x", 1:3)`: x1, x2 and x3

select()와 함께 쓸 수 있는 도우미 함수들

- `starts_with("abc")`: “abc”로 시작하는 열이름 선택
- `ends_with("xyz")`: “xyz”으로 끝나는 열이름 선택
- `contains("ijk")`: “ijk”를 포함한 열이름 선택
- `matches()`: 정규표현식을 사용하여 변수이름 선택
- `num_range("x", 1:3)`: x1, x2 and x3

rename() : 열이름 바꾸기

```
rename(flights, tail_num = tailnum)
```

everything()과 함께 select() 사용

데이터 프레임의 변수 순서를 바꿀때 유용

```
select(flights, time_hour, air_time,everything())
```

```
# A tibble: 336,776 x 19
             time_hour air_time  year month   day dep_time sched_dep_time
                <dttm>    <dbl> <int> <int> <int>    <int>          <int>
 1 2013-01-01 05:00:00      227  2013     1     1      517            515
 2 2013-01-01 05:00:00      227  2013     1     1      533            529
 3 2013-01-01 05:00:00      160  2013     1     1      542            540
 4 2013-01-01 05:00:00      183  2013     1     1      544            545
 5 2013-01-01 06:00:00      116  2013     1     1      554            600
 6 2013-01-01 05:00:00      150  2013     1     1      554            558
 7 2013-01-01 06:00:00      158  2013     1     1      555            600
 8 2013-01-01 06:00:00       53  2013     1     1      557            600
 9 2013-01-01 06:00:00      140  2013     1     1      557            600
10 2013-01-01 06:00:00      138  2013     1     1      558            600
# ... with 336,766 more rows, and 12 more variables: dep_delay <dbl>,
#   arr_time <int>, sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, distance <dbl>,
#   hour <dbl>, minute <dbl>
```

Exercises

1.Brainstorm as many ways as possible to select dep_time, dep_delay, arr_time, and arr_delay from flights.

2.What happens if you include the name of a variable multiple times in a select() call?

3.What does the one_of() function do? Why might it be helpful in conjunction with this vector?

```
vars <- c("year", "month", "day", "dep_delay", "arr_delay")
select(flights, one_of(vars))
```

4.Does the result of running the following code surprise you? How do the select helpers deal with case by default? How can you change that default?

```
select(flights, contains("TIME"))
```

새로운 변수 추가 mutate()

기존의 열에서 새로운 열을 만들 때 mutate()함수를 사용할 수 있다.

```
flights_sml <- select(flights,
                      year:day, ends_with("delay"), distance, air_tir

mutate(flights_sml,
  gain = arr_delay - dep_delay,
  speed = distance / air_time * 60
)
```

```
# A tibble: 336,776 x 9
    year month   day dep_delay arr_delay distance air_time  gain    speed
   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl>    <dbl>
 1  2013     1     1         2        11     1400      227     9 370.0441
 2  2013     1     1         4        20     1416      227    16 374.2731
 3  2013     1     1         2        33     1089      160    31 408.3750
 4  2013     1     1        -1       -18     1576      183   -17 516.7213
 5  2013     1     1        -6       -25      762      116   -19 394.1379
 6  2013     1     1        -4        12      719      150    16 287.6000
 7  2013     1     1        -5        19     1065      158    24 404.4304
 8  2013     1     1        -3       -14      229       53   -11 259.2453
 9  2013     1     1        -3        -8      944      140    -5 404.5714
```

방금 만든 열도 참조하여 새 열을 만들 수 있다.

```
mutate(flights_sml,
  gain = arr_delay - dep_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours
)
```

```
# A tibble: 336,776 x 10
    year month   day dep_delay arr_delay distance air_time  gain     hours
   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl>     <dbl>
 1  2013     1     1         2        11     1400      227     9 3.7833333
 2  2013     1     1         4        20     1416      227    16 3.7833333
 3  2013     1     1         2        33     1089      160    31 2.6666667
 4  2013     1     1        -1       -18     1576      183   -17 3.0500000
 5  2013     1     1        -6       -25      762      116   -19 1.9333333
 6  2013     1     1        -4        12      719      150    16 2.5000000
 7  2013     1     1        -5        19     1065      158    24 2.6333333
 8  2013     1     1        -3       -14      229       53   -11 0.8833333
 9  2013     1     1        -3        -8      944      140    -5 2.3333333
10  2013     1     1        -2         8      733      138    10 2.3000000
# ... with 336,766 more rows, and 1 more variables: gain_per_hour <dbl>
```

새로 만든 변수만 유지하고 나머지 변수는 없앨 경우 transmutate()를 쓴다.

```
transmute(flights,
  gain = arr_delay - dep_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours
)
```

```
# A tibble: 336,776 x 3
    gain     hours gain_per_hour
   <dbl>     <dbl>         <dbl>
 1     9 3.7833333      2.378855
 2    16 3.7833333      4.229075
 3    31 2.6666667     11.625000
 4   -17 3.0500000     -5.573770
 5   -19 1.9333333     -9.827586
 6    16 2.5000000      6.400000
 7    24 2.6333333      9.113924
 8   -11 0.8833333    -12.452830
 9    -5 2.3333333     -2.142857
10    10 2.3000000      4.347826
# ... with 336,766 more rows
```

mutate()와 함께 사용하는 유용한 함수들

- 산수 연산자: +, -, *, / , ^ . 예를 들어 air_time/60, hours*60 + minute
- 몫과 나머지: 몫(%/%) 연산자, 나머지(%%) 연산자

```
transmute(flights,
  dep_time,
  hour = dep_time %/% 100,
  minute = dep_time %% 100
)
```

```
# A tibble: 336,776 x 3
   dep_time  hour minute
      <int> <dbl>  <dbl>
 1      517     5     17
 2      533     5     33
 3      542     5     42
 4      544     5     44
 5      554     5     54
 6      554     5     54
 7      555     5     55
 8      557     5     57
 9      557     5     57
10      558     5     58
```

- 로그함수: log(), log2(), log10()
- 오프셋: leads(), lags()

```
(x <- 1:10)
```

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

```
lag(x)
```

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

```
lead(x)
```

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

lag()함수는 값의 변화를 계산하거나(x-lag(x)) 값이 변화했는지 찾아내는데 유용하다 (x!=lag(x))

```
x <- c(1,2,2,3,7,7,8,1)
x
```

```
[1] 1 2 2 3 7 7 8 1
```

```
lag(x)
```

```
[1] NA  1  2  2  3  7  7  8
```

```
x-lag(x)
```

```
[1] NA  1  0  1  4  0  1 -7
```

```
x != lag(x)
```

```
[1]    NA  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE
```

- 누적 총계: cumsum(), cumprod(), cummin(), cummax(), cummean()

```
(x <- 1:10)
```

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

```
cumsum(x)
```

```
[1]  1  3  6 10 15 21 28 36 45 55
```

```
cummean(x)
```

```
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5
```

- 논리적 비교: <, <=, > , >=, ==, !=

- 순위연산자: min_rank()

```
y <- c(1,2,2,NA,3,4)
min_rank(y)
```

```
[1]  1  2  2 NA  4  5
```

```
min_rank(desc(y))
```

```
[1]  5  3  3 NA  2  1
```

- 그 외 순위연산자

```
row_number(y)
```

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

```
dense_rank(y)
```

```
[1]  1  2  2 NA  3  4
```

```
percent_rank(y)
```

```
[1] 0.00 0.25 0.25   NA 0.75 1.00
```

Exercises

1.Currently `dep_time` and `sched_dep_time` are convenient to look at, but hard to compute with because they're not really continuous numbers. Convert them to a more convenient representation of number of minutes since midnight.

2.Compare `air_time` with `arr_time - dep_time`. What do you expect to see? What do you see? What do you need to do to fix it?

3.Compare `dep_time`, `sched_dep_time`, and `dep_delay`. How would you expect those three numbers to be related?

4.Find the 10 most delayed flights using a ranking function. How do you want to handle ties? Carefully read the documentation for `min_rank()`.

5.What does `1:3 + 1:10` return? Why?

6.What trigonometric functions does R provide?

Answers

1.Currently `dep_time` and `sched_dep_time` are convenient to look at, but hard to compute with because they're not really continuous numbers. Convert them to a more convenient representation of number of minutes since midnight.

```
flights %>%
    select(ends_with("dep_time")) %>%
    mutate(
        dep_min = dep_time %/% 100*60 + dep_time %% 100,
        sched_dep_min = sched_dep_time %/% 100*60 + sched_dep_time %%
    )
```

```
# A tibble: 336,776 x 4
   dep_time sched_dep_time dep_min sched_dep_min
      <int>          <int>   <dbl>         <dbl>
 1      517            515     317           315
 2      533            529     333           329
 3      542            540     342           340
 4      544            545     344           345
 5      554            600     354           360
 6      554            558     354           358
 7      555            600     355           360
 8      557            600     357           360
```

2.Compare `air_time` with `arr_time - dep_time`. What do you expect to see? What do you see? What do you need to do to fix it?

```
flights %>%
    mutate(time_diff = arr_time - dep_time) %>%
    select(air_time,arr_time,dep_time,time_diff)
```

```
# A tibble: 336,776 x 4
   air_time arr_time dep_time time_diff
      <dbl>    <int>    <int>     <int>
 1      227      830      517       313
 2      227      850      533       317
 3      160      923      542       381
 4      183     1004      544       460
 5      116      812      554       258
 6      150      740      554       186
 7      158      913      555       358
 8       53      709      557       152
 9      140      838      557       281
10      138      753      558       195
# ... with 336,766 more rows
```

```r
time2min=function(time){
    time %/%100*60 + time %%100
}
flights %>%
    mutate(
        arr_min=time2min(arr_time),
        dep_min=time2min(dep_time),
        timediff=arr_min-dep_min
    ) %>%
    select(air_time,arr_time, dep_time, timediff)
```

```
# A tibble: 336,776 x 4
   air_time arr_time dep_time timediff
      <dbl>    <int>    <int>    <dbl>
 1      227      830      517      193
 2      227      850      533      197
 3      160      923      542      221
 4      183     1004      544      260
 5      116      812      554      138
 6      150      740      554      106
 7      158      913      555      198
 8       53      709      557       72
 9      140      838      557      161
10      138      753      558      115
# ... with 336,766 more rows
```

3.Compare `dep_time`, `sched_dep_time`, and `dep_delay`. How would you expect those three numbers to be related?

```
flights %>%
    mutate(dep_diff = time2min(dep_time) - time2min(sched_dep_time))
    filter(dep_delay!=dep_diff)
```

```
# A tibble: 1,207 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      848           1835       853     1001
 2  2013     1     2       42           2359        43      518
 3  2013     1     2      126           2250       156      233
 4  2013     1     3       32           2359        33      504
 5  2013     1     3       50           2145       185      203
 6  2013     1     3      235           2359       156      700
 7  2013     1     4       25           2359        26      505
 8  2013     1     4      106           2245       141      201
 9  2013     1     5       14           2359        15      503
10  2013     1     5       37           2230       127      341
# ... with 1,197 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>, dep_diff <dbl>
```

4.Find the 10 most delayed flights using a ranking function. How do you want to handle ties? Carefully read the documentation for `min_rank()`.

```
flights %>%
    mutate(rank=min_rank(desc(arr_delay))) %>%
    arrange(desc(arr_delay)) %>%
    filter(rank<=10)
```

```
# A tibble: 10 x 20
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013     1     9      641            900      1301     1242
 2  2013     6    15     1432           1935      1137     1607
 3  2013     1    10     1121           1635      1126     1239
 4  2013     9    20     1139           1845      1014     1457
 5  2013     7    22      845           1600      1005     1044
 6  2013     4    10     1100           1900       960     1342
 7  2013     3    17     2321            810       911      135
 8  2013     7    22     2257            759       898      121
 9  2013    12     5      756           1700       896     1058
10  2013     5     3     1133           2055       878     1250
# ... with 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>, rank <int>
```

```
flights %>%
    top_n(10,arr_delay) %>%
    arrange(desc(arr_delay))
```

```
# A tibble: 10 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013     1     9      641            900      1301     1242
 2  2013     6    15     1432           1935      1137     1607
 3  2013     1    10     1121           1635      1126     1239
 4  2013     9    20     1139           1845      1014     1457
 5  2013     7    22      845           1600      1005     1044
 6  2013     4    10     1100           1900       960     1342
 7  2013     3    17     2321            810       911      135
 8  2013     7    22     2257            759       898      121
 9  2013    12     5      756           1700       896     1058
10  2013     5     3     1133           2055       878     1250
# ... with 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>
```

5.What does `1:3 + 1:10` return? Why?

```
1:3 + 1:10
```

```
 [1]  2  4  6  5  7  9  8 10 12 11
```

6.What trigonometric functions does R provide?

그룹별 요약 summarise()

summarise()함수는 데이터프레임을 요약하여 한줄로 만들어준다.

```
summarise(flights, delay = mean(dep_delay, na.rm = TRUE))
```

```
# A tibble: 1 x 1
     delay
     <dbl>
1 12.63907
```

group_by() 함수를 사용하면 데이터프레임 전체가 아니라 그룹별로 나뉜 데이터 그룹별로 함수가 적용된다. 예를 들어 날짜 별로 이륙지연 평균을 구해보면 다음과 같다.

```
by_day <- group_by(flights, year, month, day)
summarise(by_day, delay= mean(dep_delay, na.rm = TRUE))
```

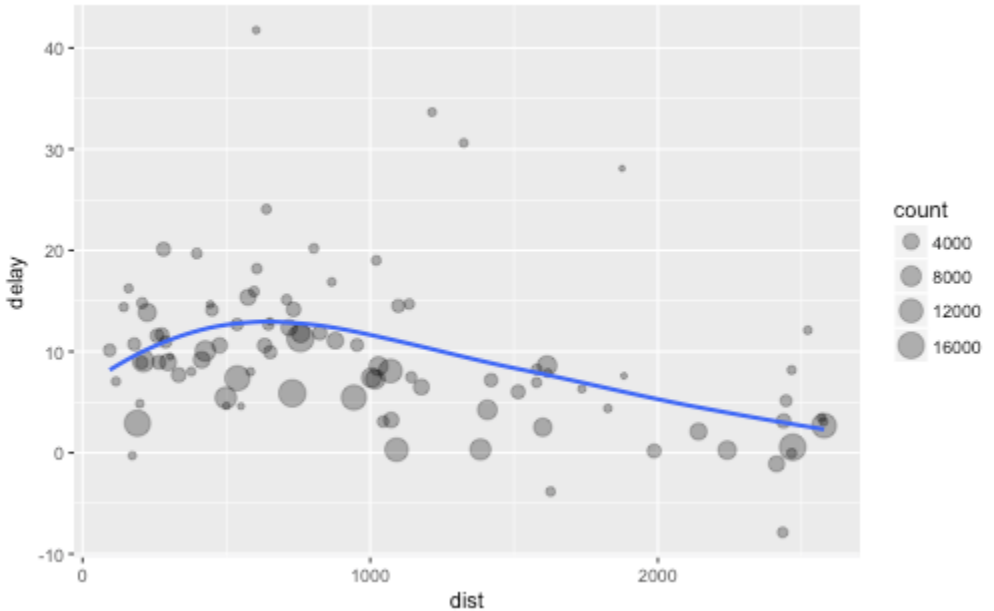
```
# A tibble: 365 x 4
# Groups:   year, month [?]
    year month   day     delay
   <int> <int> <int>     <dbl>
 1  2013     1     1 11.548926
 2  2013     1     2 13.858824
 3  2013     1     3 10.987832
 4  2013     1     4  8.951595
 5  2013     1     5  5.732218
 6  2013     1     6  7.148014
 7  2013     1     7  5.417204
 8  2013     1     8  2.553073
 9  2013     1     9  2.276477
10  2013     1    10  2.844995
# ... with 355 more rows
```

여러 작업을 하나로 연결하는 pipe

flights 데이터에서 목적지에 따른 비행거리와 도착지연과의 관계를 그림으로 나타내보자.

```
by_dest <- group_by(flights, dest)
delay <- summarise(by_dest,
  count = n(),
  dist = mean(distance, na.rm = TRUE),
  delay = mean(arr_delay, na.rm = TRUE)
)
delay <- filter(delay, count > 20, dest != "HNL")
```

```
ggplot(data = delay, mapping = aes(x = dist, y = delay)) +
  geom_point(aes(size = count), alpha = 1/3) +
  geom_smooth(se = FALSE)
```



이 그래프를 얻기 위해 다음과 같은 단계를 거쳤다.

1.먼저 목적지 별로 그룹으로 나누었다.

2.비행횟수, 도착거리 평균, 도착지연 평균을 계산하였다.

3.건수가 작은 도착지와 호놀룰루를 제외하였다.

하지만 이 경우 중간단계의 데이터프레임에 일일이 이름을 붙였다. 이 이름은 단지 다음 단계로 넘어가기 위한 이름이고 달리 쓰이는 곳도 없다.

파이프 연산자 %>%를 쓰면 다음과 같다.

```
delays <- flights %>%
  group_by(dest) %>%
  summarise(
    count = n(),
    dist = mean(distance, na.rm = TRUE),
    delay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  filter(count > 20, dest != "HNL")
```

참고) %>% 는 then으로 읽는다.

파이프 연산자

- `x %>% f(y)` turns into `f(x,y)`
- `x %>% f(y) %>% g(z)` turns into `g(f(x, y), z)`

누락치 제거

평균과 같은 집계함수는 누락치가 하나라도 있으면 누락치로 계산된다.

```
flights %>%
  group_by(year, month, day) %>%
  summarise(mean = mean(dep_delay))
```

```
# A tibble: 365 x 4
# Groups:   year, month [?]
    year month   day  mean
   <int> <int> <int> <dbl>
 1  2013     1     1    NA
 2  2013     1     2    NA
 3  2013     1     3    NA
 4  2013     1     4    NA
 5  2013     1     5    NA
 6  2013     1     6    NA
 7  2013     1     7    NA
 8  2013     1     8    NA
 9  2013     1     9    NA
10  2013     1    10    NA
# ... with 355 more rows
```

따라서 평균을 계산할때 누락치를 제거하려면 na.rm = TRUE 옵션을 준다.

```
flights %>%
  group_by(year, month, day) %>%
  summarise(mean = mean(dep_delay, na.rm=TRUE))
```

```
# A tibble: 365 x 4
# Groups:   year, month [?]
    year month   day      mean
   <int> <int> <int>     <dbl>
 1  2013     1     1 11.548926
 2  2013     1     2 13.858824
 3  2013     1     3 10.987832
 4  2013     1     4  8.951595
 5  2013     1     5  5.732218
 6  2013     1     6  7.148014
 7  2013     1     7  5.417204
 8  2013     1     8  2.553073
 9  2013     1     9  2.276477
10  2013     1    10  2.844995
# ... with 355 more rows
```

누락치를 제거한 데이터를 not_cancelled 에 저장한다.

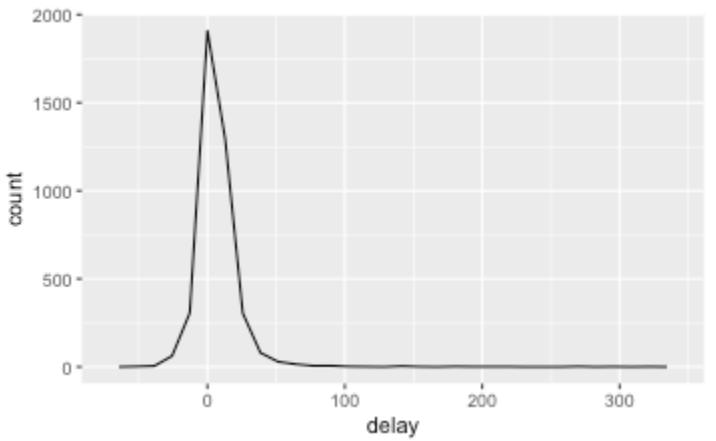
```
not_cancelled <- flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay))

not_cancelled %>%
  group_by(year, month, day) %>%
  summarise(mean = mean(dep_delay))
```

```
# A tibble: 365 x 4
# Groups:   year, month [?]
    year month   day      mean
   <int> <int> <int>     <dbl>
 1  2013     1     1 11.435620
 2  2013     1     2 13.677802
 3  2013     1     3 10.907778
 4  2013     1     4  8.965859
 5  2013     1     5  5.732218
 6  2013     1     6  7.145959
 7  2013     1     7  5.417204
 8  2013     1     8  2.558296
 9  2013     1     9  2.301232
10  2013     1    10  2.844995
# ... with 355 more rows
```

Counts

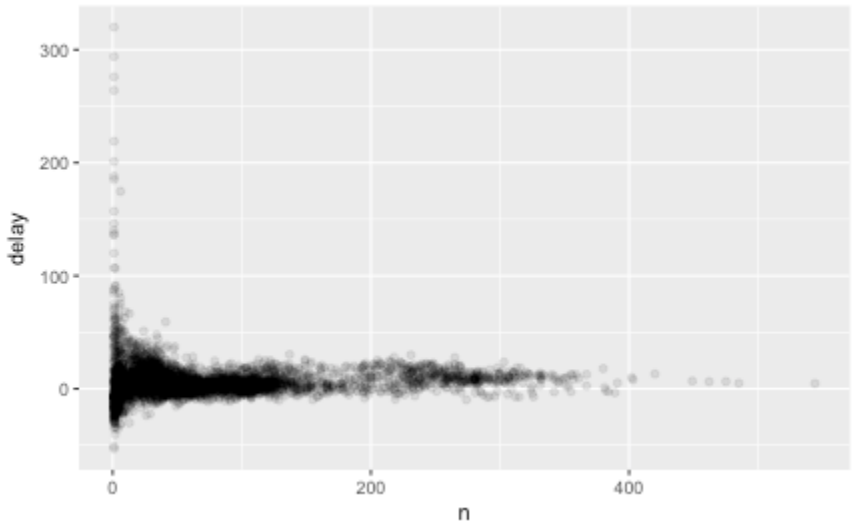
집계를 할때 count(n()) 또는 누락되지 않은 count(sum(!is.na(x))) 를 포함하는 것이 좋다. 결론을 내리기 전에 적은 양의 데이터에 의해 결론이 내려지지 않는지 count를 살펴보아야 한다. 예를 들어 평균 도착지연이 가장 많은 비행기(tail number)가 어떤 것인지 살펴보자.

```
delays <- not_cancelled %>%
  group_by(tailnum) %>%
  summarise(delay = mean(arr_delay))

ggplot(data = delays, mapping = aes(x = delay)) +
  geom_freqpoly()
```



비행횟수와 평균도착지연 관계를 산점도로 나타내보면 다음과 같다.

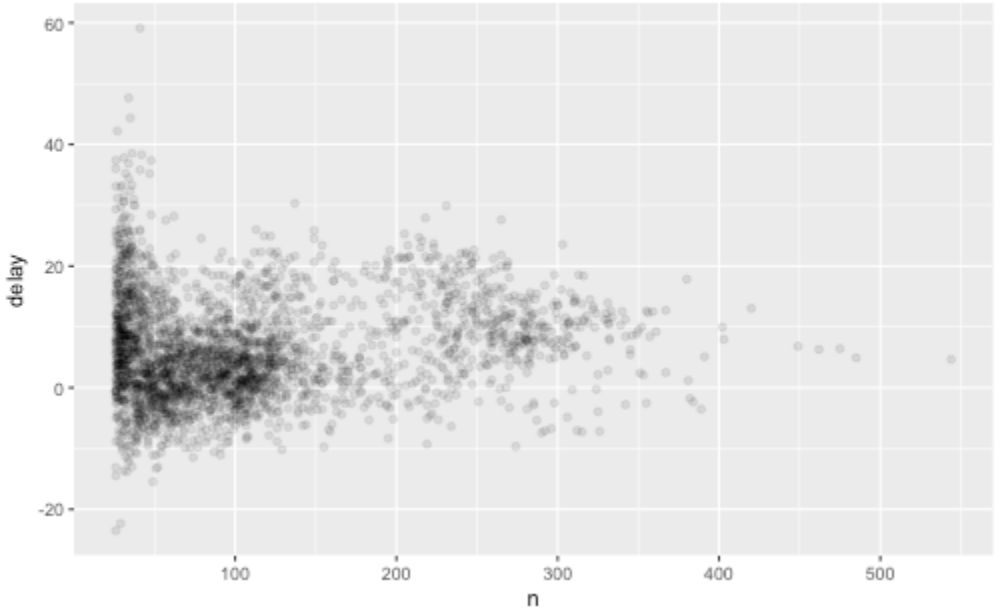
```
delays <- not_cancelled %>%
  group_by(tailnum) %>%
  summarise(
    delay=mean(arr_delay, na.rm = TRUE),
    n = n()
  )
ggplot(data = delays, mapping = aes(x = n, y = delay)) +
  geom_point(alpha = 1/10)
```



비행횟수가 적은 경우 도착지연시간의 변동이 크다는 것을 알수 있다.

이런 경우 관찰치가 적은 그룹을 필터링해서 걸러내는 것이 유용할 수 있다.

```
delays %>%
  filter(n > 25) %>%
  ggplot(mapping = aes(x = n, y = delay)) +
    geom_point(alpha = 1/10)
```



타수와 타율의 관계

Lahman패키지의 Batting 데이타를 사용하여 타수와 타율의 관계를 살펴보자.
Lahman패키지가 설치되어 있지 않은 경우 설치한다.

```
install.packages("Lahman")
```

타자의 타율(batting average, ba) 은 안타수를 타수로 나누어 계산한다.

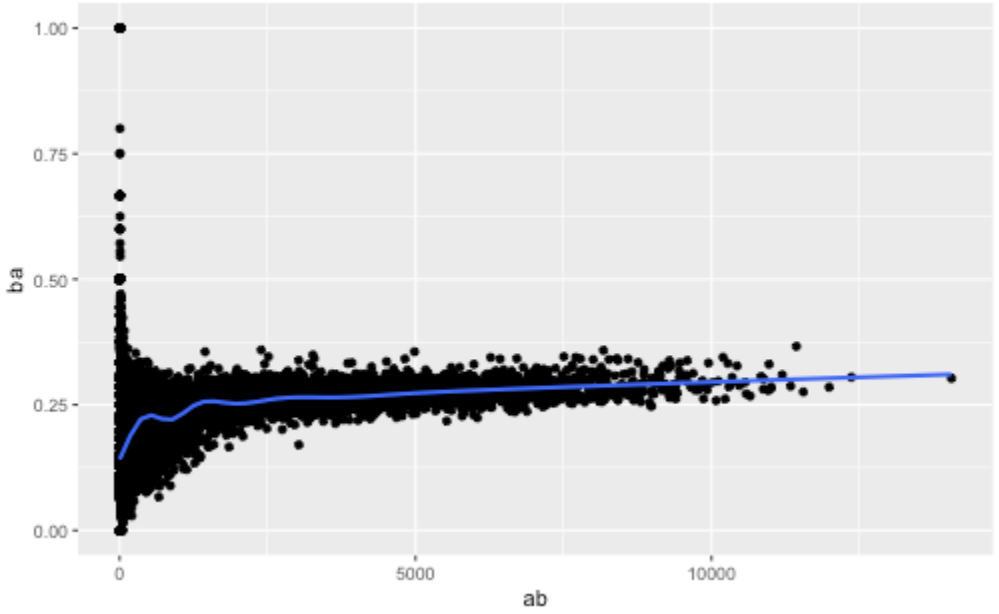
```
batting <- as_tibble(Lahman::Batting)

batters <- batting %>%
  group_by(playerID) %>%
  summarise(
    ba = sum(H, na.rm = TRUE) / sum(AB, na.rm = TRUE),
    ab = sum(AB, na.rm = TRUE)
  )
```

타율이 가장 높은 10명을 보면 다음과 같다.

```
batters %>%
    top_n(10,ba)
```

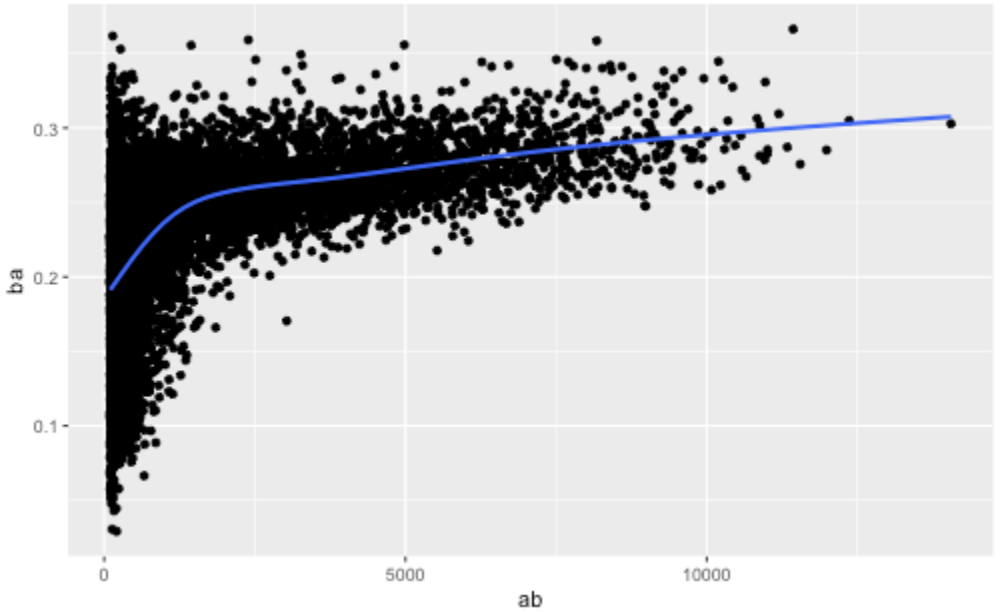
```
# A tibble: 91 x 3
    playerID    ba    ab
       <chr> <dbl> <int>
 1 abramge01     1     1
 2 banisje01     1     1
 3 bartocl01     1     1
 4  bassdo01     1     1
 5 berrijo01     1     1
 6 birasst01     1     2
 7 bruneju01     1     1
 8 burnscb01     1     1
 9 cammaer01     1     1
10  campsh01     1     1
# ... with 81 more rows
```

```
batters %>%
  ggplot(mapping = aes(x = ab, y = ba)) +
    geom_point() +
    geom_smooth(se = FALSE)
```



100타수 이상인 타자들에 대해서 산점도를 그려보면 다음과 같다.

```
batters %>%
  filter(ab > 100) %>%
  ggplot(mapping = aes(x = ab, y = ba)) +
    geom_point() +
    geom_smooth(se = FALSE)
```



유용한 집계함수

- 평균, 중앙값: mean(), median()

```
not_cancelled %>%
  group_by(year, month, day) %>%
  summarise(
    avg_delay1 = mean(arr_delay),
    avg_delay2 = mean(arr_delay[arr_delay > 0]) # the average positiv
  )
```

```
# A tibble: 365 x 5
# Groups:   year, month [?]
    year month   day avg_delay1 avg_delay2
   <int> <int> <int>      <dbl>      <dbl>
 1  2013     1     1 12.6510229   32.48156
 2  2013     1     2 12.6928879   32.02991
 3  2013     1     3  5.7333333   27.66087
 4  2013     1     4 -1.9328194   28.30976
 5  2013     1     5 -1.5258020   22.55882
 6  2013     1     6  4.2364294   24.37270
 7  2013     1     7 -4.9473118   27.76132
 8  2013     1     8 -3.2275785   20.78909
 9  2013     1     9 -0.2642777   25.63415
10  2013     1    10 -5.8988159   27.34545
```

- 퍼진 정도: sd(), IQR(), mad()

```
not_cancelled %>%
  group_by(dest) %>%
  summarise(distance_sd = sd(distance)) %>%
  arrange(desc(distance_sd))
```

```
# A tibble: 104 x 2
    dest distance_sd
   <chr>       <dbl>
 1   EGE   10.542765
 2   SAN   10.350094
 3   SFO   10.216017
 4   HNL   10.004197
 5   SEA    9.977993
 6   LAS    9.907786
 7   PDX    9.873299
 8   PHX    9.862546
 9   LAX    9.657195
10   IND    9.458066
# ... with 94 more rows
```

- 순위: min(x), quantile(x,0.25), max(x)
- 순서 : first(x), nth(x), last(x)
- Counts : n(), sum(!is.na(x)), n_distinct(x)

```
# Which destinations have the most carriers?
not_cancelled %>%
  group_by(dest) %>%
  summarise(carriers = n_distinct(carrier)) %>%
  arrange(desc(carriers))
```

```
# A tibble: 104 x 2
    dest carriers
   <chr>    <int>
 1   ATL        7
 2   BOS        7
 3   CLT        7
 4   ORD        7
 5   TPA        7
 6   AUS        6
 7   DCA        6
 8   DTW        6
 9   IAD        6
10   MSP        6
# ... with 94 more rows
```

- count()

```
not_cancelled %>%
  count(dest)
```

```
# A tibble: 104 x 2
    dest     n
   <chr> <int>
 1   ABQ   254
 2   ACK   264
 3   ALB   418
 4   ANC     8
 5   ATL 16837
 6   AUS  2411
 7   AVL   261
 8   BDL   412
 9   BGR   358
10   BHM   269
# ... with 94 more rows
```

- count() for sum

```
not_cancelled %>%
  count(tailnum, wt = distance)
```

```
# A tibble: 4,037 x 2
   tailnum      n
     <chr>  <dbl>
 1  D942DN   3418
 2  N0EGMQ 239143
 3  N10156 109664
 4  N102UW  25722
 5  N103US  24619
 6  N104UW  24616
 7  N10575 139903
 8  N105UW  23618
 9  N107US  21677
10  N108UW  32070
# ... with 4,027 more rows
```

- counts for logical values : sum(x>10), mean(x==0)

논리값을 숫자함수로 다룰때 TRUE는 1로 FALSE는 0으로 바뀐다. 따라서 sum(x)는 TRUE의 갯수, mean(x)는 비율이 된다. 예를 들어 날짜별로 새벽 5시 전에 출발한 비행기의 대수와 비율을 보면 다음과 같다.

```
not_cancelled %>%
    group_by(year,month,day) %>%
    summarise(
        n_early=sum(dep_time<=500),
        early_perc=mean(dep_time<=500)
    )
```

```
# A tibble: 365 x 5
# Groups:   year, month [?]
    year month   day n_early  early_perc
   <int> <int> <int>   <int>       <dbl>
 1  2013     1     1       0 0.000000000
 2  2013     1     2       3 0.003232759
 3  2013     1     3       4 0.004444444
 4  2013     1     4       3 0.003303965
 5  2013     1     5       3 0.004184100
 6  2013     1     6       2 0.002412545
 7  2013     1     7       2 0.002150538
 8  2013     1     8       1 0.001121076
 9  2013     1     9       3 0.003359462
10  2013     1    10       3 0.003229279
```

날짜별로 1시간 이상 연착된 비행기의 비율은?

```
not_cancelled %>%
    group_by(year,month,day) %>%
    summarise(
        hour_perc=mean(arr_delay>60)
    )
```

```
# A tibble: 365 x 4
# Groups:   year, month [?]
    year month   day  hour_perc
   <int> <int> <int>      <dbl>
 1  2013     1     1 0.07220217
 2  2013     1     2 0.08512931
 3  2013     1     3 0.05666667
 4  2013     1     4 0.03964758
 5  2013     1     5 0.03486750
 6  2013     1     6 0.04704463
 7  2013     1     7 0.03333333
 8  2013     1     8 0.02130045
 9  2013     1     9 0.02015677
10  2013     1    10 0.01829925
# ... with 355 more rows
```

Exercises

1.Brainstorm at least 5 different ways to assess the typical delay characteristics of a group of flights. Consider the following scenarios:

- A flight is 15 minutes early 50% of the time, and 15 minutes late 50% of the time.
- A flight is always 10 minutes late.
- A flight is 30 minutes early 50% of the time, and 30 minutes late 50% of the time.
- 99% of the time a flight is on time. 1% of the time it's 2 hours late.

Which is more important: arrival delay or departure delay?

2.Come up with another approach that will give you the same output as not_cancelled %>% count(dest) and not_cancelled %>% count(tailnum, wt = distance) (without using count()).

3.Our definition of cancelled flights (is.na(dep_delay) | is.na(arr_delay)) is slightly suboptimal. Why? Which is the most important column?

4.Look at the number of cancelled flights per day. Is there a pattern? Is the proportion of cancelled flights related to the average delay?

Grouped filters and mutates

- Find worst members od each group

```
flights_sml %>%
  group_by(year, month, day) %>%
  filter(rank(desc(arr_delay)) < 2)
```

```
# A tibble: 366 x 7
# Groups:   year, month, day [365]
    year month   day dep_delay arr_delay distance air_time
   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>
 1  2013     1     1       853       851      184       41
 2  2013     1     2       337       368     2586      346
 3  2013     1     3       291       285      765      125
 4  2013     1     4       288       276      708      109
 5  2013     1     5       327       308     1010      158
 6  2013     1     6       163       175     1167      200
 7  2013     1     7       366       368     1076      153
 8  2013     1     8       188       184      229       43
 9  2013     1     9      1301      1272     4983      640
10  2013     1    10      1126      1109      719      111
# ... with 356 more rows
```

```
  #top_n(1,arr_delay)
```

Exercises

1.Refer back to the lists of useful mutate and filtering functions. Describe how each operation changes when you combine it with grouping.

2.Which plane (tailnum) has the worst on-time record?

3.What time of day should you fly if you want to avoid delays as much as possible?

4.For each destination, compute the total minutes of delay. For each, flight, compute the proportion of the total delay for its destination.

5.Delays are typically temporally correlated: even once the problem that caused the initial delay has been resolved, later flights are delayed to allow earlier flights to leave. Using lag() explore how the delay of a flight is related to the delay of the immediately preceding flight.

6.Look at each destination. Can you find flights that are suspiciously fast? (i.e. flights that represent a potential data entry error). Compute the air time a flight relative to the shortest flight to that destination. Which flights were most delayed in the air?

7.Find all destinations that are flown by at least two carriers. Use that information to rank the carriers.