week_5_veri_isleme

Hakan Mehmetcik

2024-10-31

```
Warning: package 'tidyverse' was built under R version 4.3.3
```

```
Warning: package 'tidyr' was built under R version 4.3.3
```

```
Warning: package 'dplyr' was built under R version 4.3.3
```

```
Warning: package 'lubridate' was built under R version 4.3.3
```

```
Warning: package 'nycflights13' was built under R version 4.3.3
```

Veri Düzenleme için Bir Dilbilgisi

Tıpkı `ggplot2` paketinin veri görselleştirme için güçlü ve esnek bir dilbilgisi sunması gibi, `dplyr` paketi de R programlama dilinde veri manipülasyonu ve düzenleme işlemleri için son derece kullanışlı ve etkili bir dilbilgisi sunar. Bu paket, veri analizi sürecini daha verimli ve anlaşılır hale getiren bir dizi araç sağlar.

Özellikle, `dplyr` paketi veri çerçeveleri üzerinde çeşitli karmaşık işlemler gerçekleştirmek için "fiil" olarak adlandırılan işlevsel fonksiyonlar sunar. Bu fiiller, veri setlerini filtreleme, sıralama, gruplama, özetleme ve birleştirme gibi temel veri manipülasyon görevlerini gerçekleştirmek için kullanılır. Her bir fiil, belirli bir görevi yerine getirmek üzere tasarlanmıştır ve birbirleriyle kolayca birleştirilebilir, böylece karmaşık veri dönüşümlerini bile anlaşılır ve okunabilir bir şekilde ifade etmek mümkün olur.

> **Note**
>
> `dplyr` paketindeki temel fiiller şunlardır:
>
> - `select()`: Veri çerçevesinden belirli sütunları seçmek için kullanılır.

- `filter()`: Belirli koşulları sağlayan satırları seçmek için kullanılır.
- `mutate()`: Yeni sütunlar oluşturmak veya mevcut sütunları değiştirmek için kullanılır.
- `arrange()`: Veri çerçevesindeki satırları belirli sütunlara göre sıralamak için kullanılır.
- `summarize()`: Verileri satırlar arasında özetlemek için kullanılır.

Bu fiilleri birleştirerek karmaşık veri manipülasyon işlemlerini kolayca gerçekleştirebilirsiniz. `dplyr` paketi, R ile veri analizi yapan herkes için olmazsa olmaz bir araçtır. Bu paketin sunduğu dilbilgisi yaklaşımı, veri manipülasyonunu daha kolay, anlaşılır ve etkili hale getirir.

1. select() ve filter()

- **`select()`**: Veri çerçevesinden belirli **sütunları** seçmek için kullanılır. Örneğin, `select(veri, sutun1, sutun2)` kodu, `veri` adlı veri çerçevesinden sadece `sutun1` ve `sutun2` adlı sütunları seçer. Bu fonksiyon, veri setinizi sadeleştirmek ve sadece ihtiyacınız olan değişkenleri tutmak için faydalıdır.
- **`filter()`**: Belirli koşulları sağlayan **satırları** seçmek için kullanılır. Örneğin, `filter(veri, sutun1 > 10)` kodu, `veri` veri çerçevesinde `sutun1` değerleri 10'dan büyük olan satırları filtreler. Bu fonksiyon, analiziniz için belirli kriterleri karşılayan gözlemleri seçmenize olanak tanır.

```
presidential |>
  select(name, party)
```

```
# A tibble: 12 x 2
   name       party
   <chr>      <chr>
 1 Eisenhower Republican
 2 Kennedy    Democratic
 3 Johnson    Democratic
 4 Nixon      Republican
 5 Ford       Republican
 6 Carter     Democratic
 7 Reagan     Republican
 8 Bush       Republican
 9 Clinton    Democratic
```

```
10 Bush       Republican
11 Obama      Democratic
12 Trump      Republican
```

> **Note**
>
> `tidyverse` içindeki `pipe` operatörü (`|>`) veri manipülasyonunu ve analizini kolaylaştırmak için kullanılır. Kodunuzu daha okunabilir ve anlaşılır hale getirir.
> **`pipe` Operatörü Nasıl Çalışır?**
> `pipe` operatörü (`|>`), bir fonksiyonun çıktısını bir sonraki fonksiyonun girdisi olarak kullanmanıza olanak tanır. Bu sayede, iç içe geçmiş fonksiyonlar yerine, işlemleri adım adım zincirleme bir şekilde yazabilirsiniz.

```
presidential |>
  filter(party=="Republican")
```

```
# A tibble: 7 x 4
  name       start      end        party
  <chr>      <date>     <date>     <chr>
1 Eisenhower 1953-01-20 1961-01-20 Republican
2 Nixon      1969-01-20 1974-08-09 Republican
3 Ford       1974-08-09 1977-01-20 Republican
4 Reagan     1981-01-20 1989-01-20 Republican
5 Bush       1989-01-20 1993-01-20 Republican
6 Bush       2001-01-20 2009-01-20 Republican
7 Trump      2017-01-20 2021-01-20 Republican
```

`select` ve `filter`'ı birlikte kullanmaya ne dersiniz?

```
presidential |>
  select(name, party) |>
  filter(party=="Republican")
```

```
# A tibble: 7 x 2
  name       party
  <chr>      <chr>
1 Eisenhower Republican
2 Nixon      Republican
3 Ford       Republican
4 Reagan     Republican
```

```
5 Bush       Republican
6 Bush       Republican
7 Trump      Republican
```

`filter()` ve `select()` komutlarını birleştirmek, çok belirli bilgilere ulaşmayı sağlar. Örneğin, presidential veri setinde Watergate skandalından beri görev yapan Demokrat başkanları bulabiliriz.

```
presidential |>
  filter(year(start) > 1973 & party == "Democratic") |>
  select(name)
```

```
# A tibble: 3 x 1
  name
  <chr>
1 Carter
2 Clinton
3 Obama
```

2. mutate() ve rename()

Veri analizi yaparken sıklıkla verilerimiz üzerinde değişiklikler yapmamız gerekir. Bu değişiklikler, yeni değişkenler oluşturmayı, mevcut değişkenleri yeniden tanımlamayı veya değişkenlerin isimlerini değiştirmeyi içerebilir. `dplyr` paketi, `mutate()` ve `rename()` fonksiyonları ile bu işlemleri kolayca yapmamızı sağlar.

Örneğin, elimizde ABD başkanlarının görev sürelerine ilişkin bir veri setimiz olan presidential verisini düşünelim. Bu veri setinde, her başkanın göreve başlama ve görevden ayrılma tarihleri bulunmaktadır. Ancak, her başkanın görev süresinin uzunluğunu gösteren bir değişken yoktur. `mutate()` fonksiyonunu kullanarak, bu bilgiyi mevcut tarihlerden hesaplayabilir ve veri setimize yeni bir sütun olarak ekleyebiliriz.

> **Note**
>
> Mevcut veri setini değiştirmek yerine, genellikle yeni bir veri seti oluşturmak iyi bir uygulamadır. Bu sayede, orijinal veri setini koruyabilir ve üzerinde istediğiniz değişiklikleri yapabilirsiniz. Örneğin, `mutate()` fonksiyonunun çıktısını yeni bir veri setine kaydedebilirsiniz.

```
my_presidents <- presidential |>
  mutate(term.length = interval(start, end) / dyears(1))
```

```r
head(my_presidents)
```

```
# A tibble: 6 x 5
  name       start      end        party      term.length
  <chr>      <date>     <date>     <chr>            <dbl>
1 Eisenhower 1953-01-20 1961-01-20 Republican        8
2 Kennedy    1961-01-20 1963-11-22 Democratic        2.84
3 Johnson    1963-11-22 1969-01-20 Democratic        5.16
4 Nixon      1969-01-20 1974-08-09 Republican        5.55
5 Ford       1974-08-09 1977-01-20 Republican        2.45
6 Carter     1977-01-20 1981-01-20 Democratic        4
```

`mutate()` fonksiyonu, mevcut bir sütundaki verileri değiştirmek için de kullanılabilir. Örneğin, veri çerçevemize her başkanın seçildiği yılı içeren bir değişken eklemek istediğimizi düşünelim. İlk (basit) yaklaşımımız, her başkanın göreve başlamadan önceki yıl seçildiğini varsayabilir. `mutate()` fonksiyonunun bir veri çerçevesi döndürdüğünü unutmayın. Bu nedenle, mevcut veri çerçevemizi değiştirmek istiyorsak, onu yeni sonuçlarla güncellememiz gerekir.

```r
my_presidents <- my_presidents |>
  mutate(elected = year(start) - 1)
head(my_presidents)
```

```
# A tibble: 6 x 6
  name       start      end        party      term.length elected
  <chr>      <date>     <date>     <chr>            <dbl>   <dbl>
1 Eisenhower 1953-01-20 1961-01-20 Republican        8       1952
2 Kennedy    1961-01-20 1963-11-22 Democratic        2.84    1960
3 Johnson    1963-11-22 1969-01-20 Democratic        5.16    1962
4 Nixon      1969-01-20 1974-08-09 Republican        5.55    1968
5 Ford       1974-08-09 1977-01-20 Republican        2.45    1973
6 Carter     1977-01-20 1981-01-20 Democratic        4       1976
```

`rename()` fonksiyonu ise, değişkenlerin isimlerini değiştirmek için kullanılır.

```r
my_presidents <- my_presidents |>
  rename(term_length = term.length)
head(my_presidents)
```

```
# A tibble: 6 x 6
```

```
  name       start      end        party      term_length elected
  <chr>      <date>     <date>     <chr>            <dbl>   <dbl>
1 Eisenhower 1953-01-20 1961-01-20 Republican        8       1952
2 Kennedy    1961-01-20 1963-11-22 Democratic        2.84    1960
3 Johnson    1963-11-22 1969-01-20 Democratic        5.16    1962
4 Nixon      1969-01-20 1974-08-09 Republican        5.55    1968
5 Ford       1974-08-09 1977-01-20 Republican        2.45    1973
6 Carter     1977-01-20 1981-01-20 Democratic        4       1976
```

Note

`janitor` paketi, R ile veri temizleme işlemlerini kolaylaştırmak için geliştirilmiş kullanışlı bir pakettir. Özellikle değişken isimlerini temizlemek ve düzenlemek için sunduğu `clean_names()` fonksiyonu oldukça faydalıdır.

`clean_names()` fonksiyonu ne işe yarar?

`clean_names()` fonksiyonu, veri setinizdeki değişken isimlerini otomatik olarak temizler ve standart bir formata dönüştürür. Bu sayede, değişken isimleriyle çalışmak daha kolay hale gelir ve kodunuz daha okunabilir olur.

`clean_names()` fonksiyonu hangi işlemleri yapar?

- **Büyük/küçük harf dönüşümü:** Tüm harfleri küçük harfe dönüştürür.
- **Boşlukların kaldırılması:** Değişken isimlerindeki boşlukları kaldırır veya alt çizgi (_) ile değiştirir.
- **Özel karakterlerin kaldırılması:** Noktalama işaretleri ve diğer özel karakterleri kaldırır veya alt çizgi (_) ile değiştirir.
- **Sayıların eklenmesi:** Değişken isimlerinin başına veya sonuna sayılar ekler (eğer aynı isimde birden fazla değişken varsa).
- **Standart formata dönüştürme:** Değişken isimlerini "snake_case" (alt çizgi ile ayrılmış küçük harfli kelimeler) gibi standart bir formata dönüştürür.

`clean_names()` fonksiyonu nasıl kullanılır?

`clean_names()` fonksiyonunu kullanmak için öncelikle `janitor` paketini yüklemeniz gerekir:
install.packages("janitor")
library(janitor)

3. arrange()

Temel bir R fonksiyonu olan `sort()` fonksiyonu bir vektörü sıralar, ancak bir veri çerçevesini sıralamaz. `arrange()` fonksiyonu ise bir veri çerçevesini sıralar. `arrange()` fonksiyonunu bir veri çerçevesinde kullanmak için, veri çerçevesini ve sıralamak istediğiniz sütunu belirtmeniz gerekir. Ayrıca, sıralama yönünü de belirtmeniz gerekir. Birden çok sıralama koşulu belirtmek, eşitlikleri çözmeye yardımcı olur.

```
my_presidents |>
  arrange(desc(term_length))
```

```
# A tibble: 12 x 6
   name       start      end        party      term_length elected
   <chr>      <date>     <date>     <chr>            <dbl>   <dbl>
 1 Eisenhower 1953-01-20 1961-01-20 Republican        8       1952
 2 Reagan     1981-01-20 1989-01-20 Republican        8       1980
 3 Clinton    1993-01-20 2001-01-20 Democratic        8       1992
 4 Bush       2001-01-20 2009-01-20 Republican        8       2000
 5 Obama      2009-01-20 2017-01-20 Democratic        8       2008
 6 Nixon      1969-01-20 1974-08-09 Republican        5.55    1968
 7 Johnson    1963-11-22 1969-01-20 Democratic        5.16    1962
 8 Carter     1977-01-20 1981-01-20 Democratic        4       1976
 9 Bush       1989-01-20 1993-01-20 Republican        4       1988
10 Trump      2017-01-20 2021-01-20 Republican        4       2016
11 Kennedy    1961-01-20 1963-11-22 Democratic        2.84    1960
12 Ford       1974-08-09 1977-01-20 Republican        2.45    1973
```

Başkanlık veri çerçevemizi her başkanın görev süresinin uzunluğuna göre sıralamak için, term_length sütununun azalan sırada olmasını belirtiyoruz. Ayrıca, parti ve seçilme yılına göre de sıralama yapabiliriz.

```
my_presidents |>
  arrange(desc(term_length), party, elected)
```

```
# A tibble: 12 x 6
   name       start      end        party      term_length elected
   <chr>      <date>     <date>     <chr>            <dbl>   <dbl>
 1 Clinton    1993-01-20 2001-01-20 Democratic        8       1992
 2 Obama      2009-01-20 2017-01-20 Democratic        8       2008
 3 Eisenhower 1953-01-20 1961-01-20 Republican        8       1952
 4 Reagan     1981-01-20 1989-01-20 Republican        8       1980
```

```
 5 Bush       2001-01-20 2009-01-20 Republican        8       2000
 6 Nixon      1969-01-20 1974-08-09 Republican        5.55    1968
 7 Johnson    1963-11-22 1969-01-20 Democratic        5.16    1962
 8 Carter     1977-01-20 1981-01-20 Democratic        4       1976
 9 Bush       1989-01-20 1993-01-20 Republican        4       1988
10 Trump      2017-01-20 2021-01-20 Republican        4       2016
11 Kennedy    1961-01-20 1963-11-22 Democratic        2.84    1960
12 Ford       1974-08-09 1977-01-20 Republican        2.45    1973
```

4. summarize() ve group_by()

Tek tablo analizi için beş fiilimizden sonuncusu, neredeyse her zaman group_by() ile birlikte kullanılan summarize()'dır. Önceki dört fiil, bir veri çerçevesini güçlü ve esnek şekillerde işlemek için bize araçlar sağladı. Ancak yalnızca bu dört fiille gerçekleştirebileceğimiz analizlerin kapsamı sınırlıdır. group_by() ile birlikte summarize() kullanmak ise bize karşılaştırmalar yapma imkanı sunar.

```
my_presidents |>
  summarize(
    N = n(),
    first_year = min(year(start)),
    last_year = max(year(end)),
    num_dems = sum(party == "Democratic"),
    years = sum(term_length),
    avg_term_length = mean(term_length)
  )
```

```
# A tibble: 1 x 6
      N first_year last_year num_dems years avg_term_length
  <int>      <dbl>     <dbl>    <int> <dbl>           <dbl>
1    12       1953      2021        5    68            5.67
```

Sonraki iki değişken, bu başkanlardan birinin göreve başladığı ilk yılı ve en son başkanın görevinin bittiği yılı belirler. İlki, `start` sütunundaki en küçük yıl, ikincisi ise `end` sütunundaki en büyük yıldır. `num_dems` değişkeni, `party` değişkeninin "Democratic" olduğu satır sayısını sayar. Son iki değişken ise `term_length` değişkeninin toplamını ve ortalamasını hesaplar. Sonuçlar gösteriyor ki, 1953'ten 2021'e kadar görev yapan 12 başkandan 5'i Demokrat'tır ve bu 68 yıllık dönemde ortalama görev süresi yaklaşık 5,6 yıldır.

```
my_presidents |>
  group_by(party) |>
```

```
  summarize(
    N = n(),
    first_year = min(year(start)),
    last_year = max(year(end)),
    num_president = sum(party == "Democratic" | party == "Republican"),
    years = sum(term_length),
    avg_term_length = mean(term_length)
  )
```

```
# A tibble: 2 x 7
  party          N first_year last_year num_president years avg_term_length
  <chr>      <int>      <dbl>     <dbl>         <int> <dbl>           <dbl>
1 Democratic     5       1961      2017             5    28            5.6
2 Republican     7       1953      2021             7    40            5.71
```

Dplyr için bir ek örnek

Bu örnekte `dplyr`'ın temel veri işleme fiillerini keşfetmek için `nycflights13::flights` veri setini kullanacağız. Bu veri çerçevesi, 2013 yılında New York şehrinden kalkan 336.776 uçuşun tamamını içerir. Veriler ABD Ulaştırma İstatistikleri Bürosu'ndan alınmıştır ve `?flights` komutu ile dokümantasyonuna erişebilirsiniz.

1. filter()

```
head(flights)
```

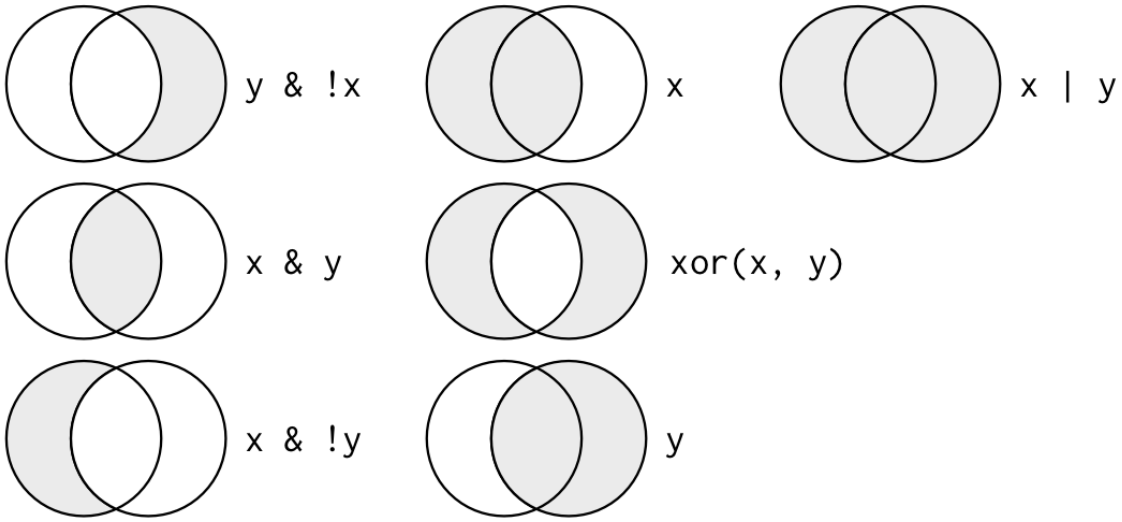
```
# A tibble: 6 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
1  2013     1     1      517            515         2      830            819
2  2013     1     1      533            529         4      850            830
3  2013     1     1      542            540         2      923            850
4  2013     1     1      544            545        -1     1004           1022
5  2013     1     1      554            600        -6      812            837
6  2013     1     1      554            558        -4      740            728
# i 11 more variables: 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>
```

```r
jan1 <- flights |>
  filter(month == 1, day == 1)
head(jan1)
```

```
# A tibble: 6 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
1  2013     1     1      517            515         2      830            819
2  2013     1     1      533            529         4      850            830
3  2013     1     1      542            540         2      923            850
4  2013     1     1      544            545        -1     1004           1022
5  2013     1     1      554            600        -6      812            837
6  2013     1     1      554            558        -4      740            728
# i 11 more variables: 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>
```

Note

R'da Kullanılan Boolean operatörler



Aşağıdaki kod, Kasım veya Aralık aylarında kalkan tüm uçuşları bulur:

```r
flights |>
  filter(month == 11 | month == 12) |>
  head()
```

```
# A tibble: 6 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
1  2013    11     1        5           2359         6      352            345
2  2013    11     1       35           2250       105      123           2356
3  2013    11     1      455            500        -5      641            651
4  2013    11     1      539            545        -6      856            827
5  2013    11     1      542            545        -3      831            855
6  2013    11     1      549            600       -11      912            923
# i 11 more variables: 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>
```

> **Note**
>
> Bu kod için kullanışlı bir kısaltma `x %in% y`'dir. Bu, `x`'in `y` içindeki değerlerden biri olduğu her satırı seçecektir. Yukarıdaki kodu yeniden yazmak için kullanabiliriz:

```
nov_dec <- filter(flights, month %in% c(11, 12))
```

De Morgan yasasını hatırlayarak karmaşık alt kümeleme işlemlerini basitleştirebilirsiniz. Bu matematiksel kural, mantıksal ifadelerin dönüşümünü sağlar ve veri analizinde oldukça kullanışlıdır. İki temel dönüşüm şu şekildedir:

1. `!(x & y)`, `!x | !y` ile aynıdır: Bu, "x ve y'nin değili" ifadesinin "x'in değili veya y'nin değili" ile eşdeğer olduğunu gösterir.
2. `!(x | y)`, `!x & !y` ile aynıdır: Bu da "x veya y'nin değili" ifadesinin "x'in değili ve y'nin değili" ile eşdeğer olduğunu belirtir.

Bu kurallar, karmaşık sorguları basitleştirmek ve daha etkili hale getirmek için kullanılabilir. Örneğin, havayolu verilerini analiz ederken, varışta veya kalkışta iki saatten fazla gecikmeyen uçuşları bulmak istiyorsanız, şu iki filtreden birini kullanabilirsiniz:

```
flights |>
  filter((arr_delay > 120 | dep_delay > 120))
```

```
# A tibble: 11,422 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
1  2013     1     1      811            630       101     1047            830
2  2013     1     1      848           1835       853     1001           1950
```

```
 3  2013     1     1      957            733       144     1056            853
 4  2013     1     1     1114            900       134     1447           1222
 5  2013     1     1     1505           1310       115     1638           1431
 6  2013     1     1     1525           1340       105     1831           1626
 7  2013     1     1     1540           1338       122     2020           1825
 8  2013     1     1     1549           1445        64     1912           1656
 9  2013     1     1     1558           1359       119     1718           1515
10  2013     1     1     1732           1630        62     2028           1825
# i 11,412 more rows
# i 11 more variables: 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 |>
  filter(arr_delay <= 120, dep_delay <= 120)
```

```
# A tibble: 316,050 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# i 316,040 more rows
# i 11 more variables: 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>
```

> **Note**
>
> `filter()` yalnızca koşulun DOĞRU olduğu satırları içerir; hem YANLIŞ hem de NA değerlerini hariç tutar. Eksik değerleri korumak istiyorsanız, bunları açıkça isteyin!

`filter()` için daha fazla alıştırma

Aşağıdaki özelliklere sahip uçuşları bulun:

- Varışta iki saat veya daha fazla gecikme olan
- Houston'a (IAH veya HOU) uçan
- United, American veya Delta tarafından işletilen
- Yaz aylarında (Temmuz, Ağustos ve Eylül) kalkan
- İki saatten fazla gecikmeli varan, ancak kalkışta gecikme olmayan
- En az bir saat gecikmeli, ancak uçuş sırasında 30 dakikadan fazla telafi eden
- Gece yarısı ile sabah 6 arasında (dahil) kalkan

Bir diğer kullanışlı `dplyr` filtreleme yardımcısı `between()`'dır. Ne işe yarar? Önceki soruları cevaplamak ve gereken kodu basitleştirmek için kullanabilir misiniz?

Kaç uçuşun `dep_time` değeri eksik? Hangi diğer değişkenler eksik? Bu satırlar neyi temsil ediyor olabilir?

2. arrange()

arrange() fonksiyonu, filter() fonksiyonuna benzer şekilde çalışır, ancak satırları seçmek yerine sıralarını değiştirir. Bu fonksiyon, sıralama işlemi için bir veri çerçevesi ve bir dizi sütun adı (veya daha karmaşık ifadeler) kullanır.

```
flights |>
  arrange(year, month, day) |>
  head()
```

```
# A tibble: 6 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
1  2013     1     1      517            515         2      830            819
2  2013     1     1      533            529         4      850            830
3  2013     1     1      542            540         2      923            850
4  2013     1     1      544            545        -1     1004           1022
5  2013     1     1      554            600        -6      812            837
6  2013     1     1      554            558        -4      740            728
# i 11 more variables: 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>
```

Azalan düzende bir sütuna göre yeniden sıralamak için `desc()` kullanın:

```
flights |>
  arrange(desc(dep_delay)) |>
  head()
```

```
# A tibble: 6 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
1  2013     1     9      641            900      1301     1242           1530
2  2013     6    15     1432           1935      1137     1607           2120
3  2013     1    10     1121           1635      1126     1239           1810
4  2013     9    20     1139           1845      1014     1457           2210
5  2013     7    22      845           1600      1005     1044           1815
6  2013     4    10     1100           1900       960     1342           2211
# i 11 more variables: 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>
```

> **Note**
>
> Eksik değerler her zaman sonda sıralanır!

arrange için daha fazla alıştırma

- Tüm eksik değerleri başa sıralamak için `arrange()` fonksiyonunu nasıl kullanabilirsiniz? (İpucu: `is.na()` kullanın).
- Uçuşları en çok gecikmiş uçuşları bulmak için sıralayın. En erken kalkan uçuşları bulun.
- En hızlı (en yüksek hız) uçuşları bulmak için uçuşları sıralayın.
- Hangi uçuşlar en uzağa gitti? Hangileri en kısa mesafeyi kat etti?

3. `select()`

Yüzlerce hatta binlerce değişken içeren veri setlerine sahip olmak nadir değildir. Bu durumda, ilk zorluk genellikle gerçekten ilgilendiğiniz değişkenlere odaklanmaktır. `select()` fonksiyonu, değişkenlerin adlarına dayalı işlemler kullanarak yararlı bir alt kümeye hızlıca odaklanmanızı sağlar.

```
flights |>
  select(year, month, day) |>
  head()
```

```
# A tibble: 6 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
```

`select()` içinde kullanabileceğiniz bir dizi yardımcı fonksiyon vardır:

- `starts_with("abc")`: "abc" ile başlayan isimleri eşleştirir.
- `ends_with("xyz")`: "xyz" ile biten isimleri eşleştirir.
- `contains("ijk")`: "ijk" içeren isimleri eşleştirir.
- `matches("(.)\\1")`: düzenli bir ifadeyle eşleşen değişkenleri seçer. Bu, tekrarlanan karakterler içeren herhangi bir değişkenle eşleşir. Düzenli ifadeler hakkında daha fazla bilgiyi diziler bölümünde öğreneceksiniz.
- `num_range("x", 1:3)`: x1, x2 ve x3 ile eşleşir.

Daha fazla ayrıntı için `?select` komutuna bakın.

> **Note**
>
> `select()` fonksiyonu değişkenleri yeniden adlandırmak için kullanılabilir, ancak açıkça belirtilmeyen tüm değişkenleri kaldırdığı için nadiren kullanışlıdır. Bunun yerine, açıkça belirtilmeyen tüm değişkenleri koruyan bir `select()` çeşidi olan `rename()` fonksiyonunu kullanın.

```
flights |>
  rename(tail_num = tailnum) |>
  head()
```

```
# A tibble: 6 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
```

```
  <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
1  2013     1     1      517            515         2      830            819
2  2013     1     1      533            529         4      850            830
3  2013     1     1      542            540         2      923            850
4  2013     1     1      544            545        -1     1004           1022
5  2013     1     1      554            600        -6      812            837
6  2013     1     1      554            558        -4      740            728
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tail_num <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

> **Note**
>
> Başka bir seçenek de `select()` fonksiyonunu `everything()` yardımcısı ile birlikte kullanmaktır. Bu, veri çerçevesinin başına taşımak istediğiniz birkaç değişkeniniz varsa kullanışlıdır.

```
flights |>
  select(time_hour, air_time, everything()) |>
  head()
```

```
# A tibble: 6 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
# i 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>
```

`select()` için daha fazla alıştırma

- `flights` veri setinden `dep_time`, `dep_delay`, `arr_time` ve `arr_delay` değişkenlerini seçmek için mümkün olduğunca çok yol düşünün.
- Bir `select()` çağrısında bir değişkenin adını birden çok kez eklerseniz ne olur?
- `any_of()` fonksiyonu ne işe yarar? Bu vektörle birlikte neden yardımcı olabilir?

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

- Aşağıdaki kodu çalıştırmanın sonucu sizi şaşırtıyor mu? `select()` yardımcıları varsayılan olarak büyük/küçük harfle nasıl başa çıkıyor? Bu varsayılanı nasıl değiştirebilirsiniz?

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

4. mutate()

`mutate()` her zaman veri setinizin sonuna yeni sütunlar ekler, bu nedenle yeni değişkenleri görebilmemiz için daha dar bir veri seti oluşturarak başlayacağız.

```
flights |>
  mutate(
    gain = dep_delay - arr_time,
    hours = air_time /60,
    gain_per_hour = gain/hours
  )|>
  head()
```

```
# A tibble: 6 x 22
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
1  2013     1     1      517            515         2      830            819
2  2013     1     1      533            529         4      850            830
3  2013     1     1      542            540         2      923            850
4  2013     1     1      544            545        -1     1004           1022
5  2013     1     1      554            600        -6      812            837
6  2013     1     1      554            558        -4      740            728
# i 14 more variables: 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>, gain <dbl>, hours <dbl>,
#   gain_per_hour <dbl>
```

```
flights_sml <- flights |>
  select(year:day,
         ends_with("delay"),
         distance,
         air_time,
         arr_time,
```

```
          dep_time) |>
  mutate(
    gain = dep_delay - arr_time,
    hours = air_time /60,
    gain_per_hour = gain/hours
  ) |>
  arrange(desc(gain_per_hour)) |>
  select(gain_per_hour, everything()) |>
  filter(gain_per_hour > 400)

head(flights_sml)
```

```
# A tibble: 4 x 12
  gain_per_hour  year month   day dep_delay arr_delay distance air_time arr_time
          <dbl> <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>    <int>
1          580.  2013     6    27       419       387      246       40       32
2          488.  2013     2    26       236       217      143       24       41
3          428.  2013     7    22       898       895      762      109      121
4          420   2013     7    22       483       463      277       47      154
# i 3 more variables: dep_time <int>, gain <dbl>, hours <dbl>
```

Eğer sadece yeni değişkenleri tutmak istiyorsanız, `transmute()` fonksiyonunu kullanın:

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

```
# A tibble: 6 x 3
   gain hours gain_per_hour
  <dbl> <dbl>         <dbl>
1    -9  3.78         -2.38
2   -16  3.78         -4.23
3   -31  2.67        -11.6
4    17  3.05          5.57
5    19  1.93          9.83
6   -16  2.5           -6.4
```

`mutate()` için daha fazla alıştırma

- Şu anda `dep_time` ve `sched_dep_time` değişkenlerine bakmak kolay, ancak gerçekte sürekli sayılar olmadıkları için hesaplamalar yapmak zor. Bunları gece yarısından itibaren geçen dakika sayısının daha uygun bir gösterimine dönüştürün.
- `air_time` ile `arr_time - dep_time` karşılaştırmasını yapın. Ne görmeyi bekliyorsunuz? Ne görüyorsunuz? Düzeltmek için ne yapmanız gerekiyor?
- `dep_time`, `sched_dep_time` ve `dep_delay` değişkenlerini karşılaştırın. Bu üç sayının nasıl ilişkili olmasını beklersiniz?
- Bir sıralama fonksiyonu kullanarak en çok gecikmeli 10 uçuşu bulun. Eşitlikleri nasıl ele almak istiyorsunuz? `min_rank()` fonksiyonunun dokümantasyonunu dikkatlice okuyun.
- `1:3 + 1:10` ne döndürür? Neden?
- R hangi trigonometrik fonksiyonları sağlar?

`5. summarise()`

Son anahtar fiil `summarise()`'dır. Bir veri çerçevesini tek bir satıra daraltır:

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

```
# A tibble: 1 x 1
  delay
  <dbl>
1  12.6
```

`summarise()` fonksiyonu, `group_by()` ile eşleştirilmedikçe çok kullanışlı değildir. Bu, analiz birimini tüm veri setinden bireysel gruplara değiştirir.

```
flights |>
  group_by(year, month, day) |>
  summarise(delay = mean(dep_delay, na.rm = T))|>
  head()
```

```
`summarise()` has grouped output by 'year', 'month'. You can override using the
`.groups` argument.
```

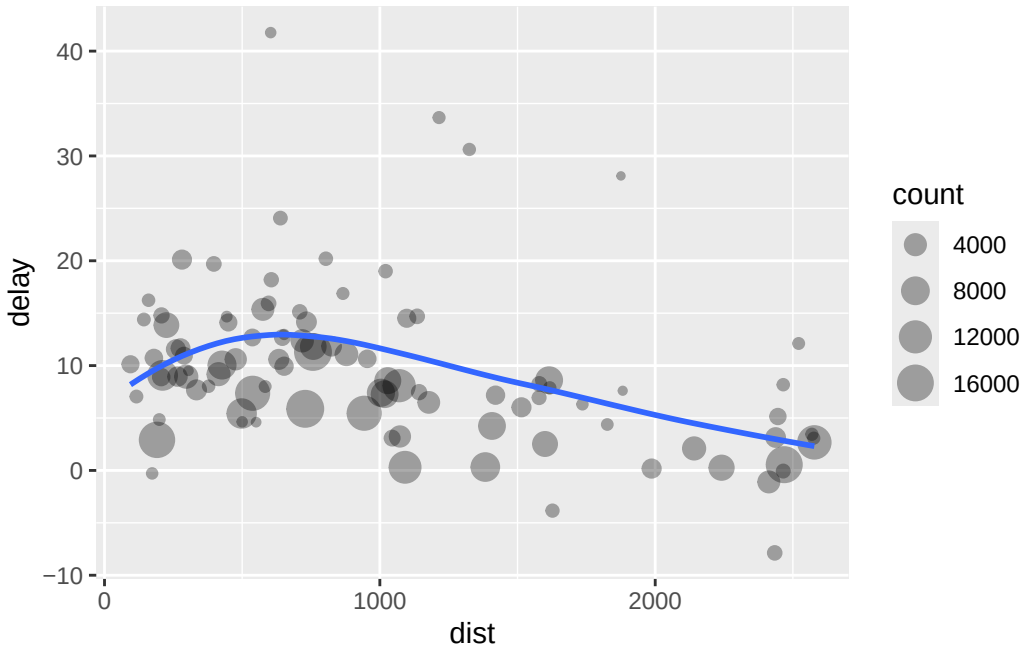
```
# A tibble: 6 x 4
# Groups:   year, month [1]
   year month   day delay
  <int> <int> <int> <dbl>
1  2013     1     1 11.5
2  2013     1     2 13.9
3  2013     1     3 11.0
4  2013     1     4  8.95
5  2013     1     5  5.73
6  2013     1     6  7.15
```

Birden çok işlemi birleştirme

Her konum için mesafe ve ortalama gecikme arasındaki ilişkiyi incelemek istediğimizi hayal edin. `dplyr` hakkında bildiklerinizi kullanarak şu şekilde bir kod yazabilirsiniz:

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

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```




```
#> `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
# It looks like delays increase with distance up to ~750 miles
# and then decrease. Maybe as flights get longer there's more
# ability to make up delays in the air?
```

Düzenli (Tidy) Veri ve R'da Veri Düzenleme

Veri düzenleme, R'da verileri "düzenli veri" (tidy data) adı verilen bir sistem kullanarak tutarlı bir şekilde organize etmeyi ifade eder. Verilerinizi bu formata getirmek başlangıçta biraz çaba gerektirir, ancak bu çaba uzun vadede karşılığını verir. Düzenli verilere ve `tidyverse` paketlerinin sağladığı düzenli araçlara sahip olduğunuzda, verileri bir gösterimden diğerine dönüştürmek için çok daha az zaman harcarsınız. Böylece, önemsediğiniz veri sorularına daha fazla zaman ayırabilirsiniz.

Bir veri setini düzenli hale getiren birbiriyle ilişkili üç kural vardır:

- Her değişken bir sütundur; her sütun bir değişkendir.
- Her gözlem bir satırdır; her satır bir gözlemdir.
- Her değer bir hücredir; her hücre tek bir değerdir.

Bu kurallar, çoğu gerçek analizin en azından biraz düzenleme gerektireceği anlamına gelir. İşe temel değişkenlerin ve gözlemlerin ne olduğunu anlayarak başlarsınız. Bu bazen kolaydır; diğer zamanlarda verileri orijinal olarak oluşturan kişilere danışmanız gerekebilir. Ardından, verilerinizi sütunlarda değişkenler ve satırlarda gözlemler olacak şekilde düzenli bir forma dönüştürürsünüz.

`tidyr` paketi, verileri dönüştürmek için iki fonksiyon sağlar: `pivot_longer()` ve `pivot_wider()`. İlk önce `pivot_longer()` ile başlayacağız çünkü en yaygın durum budur.

1. pivot_longer()

`billboard` veri seti, 2000 yılındaki şarkıların Billboard sıralamasını kaydeder:

```
head(billboard)
```

```
# A tibble: 6 x 79
  artist      track date.entered   wk1   wk2   wk3   wk4   wk5   wk6   wk7   wk8
  <chr>       <chr> <date>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 2 Pac       Baby~ 2000-02-26      87    82    72    77    87    94    99    NA
2 2Ge+her     The ~ 2000-09-02      91    87    92    NA    NA    NA    NA    NA
3 3 Doors Do~ Kryp~ 2000-04-08      81    70    68    67    66    57    54    53
4 3 Doors Do~ Loser 2000-10-21      76    76    72    69    67    65    55    59
5 504 Boyz    Wobb~ 2000-04-15      57    34    25    17    17    31    36    49
6 98^0        Give~ 2000-08-19      51    39    34    26    26    19     2     2
# i 68 more variables: wk9 <dbl>, wk10 <dbl>, wk11 <dbl>, wk12 <dbl>,
#   wk13 <dbl>, wk14 <dbl>, wk15 <dbl>, wk16 <dbl>, wk17 <dbl>, wk18 <dbl>,
#   wk19 <dbl>, wk20 <dbl>, wk21 <dbl>, wk22 <dbl>, wk23 <dbl>, wk24 <dbl>,
#   wk25 <dbl>, wk26 <dbl>, wk27 <dbl>, wk28 <dbl>, wk29 <dbl>, wk30 <dbl>,
#   wk31 <dbl>, wk32 <dbl>, wk33 <dbl>, wk34 <dbl>, wk35 <dbl>, wk36 <dbl>,
#   wk37 <dbl>, wk38 <dbl>, wk39 <dbl>, wk40 <dbl>, wk41 <dbl>, wk42 <dbl>,
#   wk43 <dbl>, wk44 <dbl>, wk45 <dbl>, wk46 <dbl>, wk47 <dbl>, wk48 <dbl>, ...
```

Bu veri setinde her bir gözlem bir şarkıdır. İlk üç sütun (`artist`, `track` ve `date.entered`) şarkıyı tanımlayan değişkenlerdir. Sonra her hafta şarkının sıralamasını tanımlayan 76 sütunumuz (`wk1-wk76`) var. Burada, sütun adları bir değişkendir (hafta) ve hücre değerleri başka bir değişkendir (sıralama).

Bu verileri düzenlemek için `pivot_longer()` fonksiyonunu kullanacağız:

```
billboard |>
  pivot_longer(
```

```
    cols = starts_with("wk"),
    names_to = "week",
    values_to = "rank",
    values_drop_na = TRUE
  )
```

```
# A tibble: 5,307 x 5
   artist  track                   date.entered week   rank
   <chr>   <chr>                   <date>       <chr> <dbl>
 1 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk1      87
 2 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk2      82
 3 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk3      72
 4 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk4      77
 5 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk5      87
 6 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk6      94
 7 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk7      99
 8 2Ge+her The Hardest Part Of ... 2000-09-02   wk1      91
 9 2Ge+her The Hardest Part Of ... 2000-09-02   wk2      87
10 2Ge+her The Hardest Part Of ... 2000-09-02   wk3      92
# i 5,297 more rows
```

Burada üç önemli argüman vardır:

- `cols`: Hangi sütunların dönüştürülmesi gerektiğini, yani hangi sütunların değişken olmadığını belirtir. Bu argüman `select()` ile aynı sözdizimini kullanır, bu nedenle burada `!c(artist, track, date.entered)` veya `starts_with("wk")` kullanabiliriz.
- `names_to`: Sütun adlarında depolanan değişkeni adlandırır, biz bu değişkene `week` adını verdik.
- `values_to`: Hücre değerlerinde depolanan değişkeni adlandırır, biz bu değişkene `rank` adını verdik.
- `values_drop_na`: `TRUE`, NA'lı satırların bırakıldığı anlamına gelir.

> **Note**
>
> Kodda "week" ve "rank" kelimelerinin tırnak içinde olduğuna dikkat edin çünkü bunlar oluşturduğumuz yeni değişkenlerdir, `pivot_longer()` çağrısını çalıştırdığımızda henüz verilerde mevcut değillerdir.

Başka bir örnek: pivot_longer()

Sütun adlarına sıkıştırılmış birden çok bilgi parçasına sahip olduğunuzda ve bunları ayrı yeni değişkenlerde depolamak istediğinizde daha zorlu bir durum ortaya çıkar. Örneğin, `table1` ve `friends` tablolarının kaynağı olan `who2` veri setini ele alalım:

```
head(who2)
```

```
# A tibble: 6 x 58
  country      year sp_m_014 sp_m_1524 sp_m_2534 sp_m_3544 sp_m_4554 sp_m_5564
  <chr>       <dbl>    <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1 Afghanistan  1980       NA        NA        NA        NA        NA        NA
2 Afghanistan  1981       NA        NA        NA        NA        NA        NA
3 Afghanistan  1982       NA        NA        NA        NA        NA        NA
4 Afghanistan  1983       NA        NA        NA        NA        NA        NA
5 Afghanistan  1984       NA        NA        NA        NA        NA        NA
6 Afghanistan  1985       NA        NA        NA        NA        NA        NA
# i 50 more variables: sp_m_65 <dbl>, sp_f_014 <dbl>, sp_f_1524 <dbl>,
#   sp_f_2534 <dbl>, sp_f_3544 <dbl>, sp_f_4554 <dbl>, sp_f_5564 <dbl>,
#   sp_f_65 <dbl>, sn_m_014 <dbl>, sn_m_1524 <dbl>, sn_m_2534 <dbl>,
#   sn_m_3544 <dbl>, sn_m_4554 <dbl>, sn_m_5564 <dbl>, sn_m_65 <dbl>,
#   sn_f_014 <dbl>, sn_f_1524 <dbl>, sn_f_2534 <dbl>, sn_f_3544 <dbl>,
#   sn_f_4554 <dbl>, sn_f_5564 <dbl>, sn_f_65 <dbl>, ep_m_014 <dbl>,
#   ep_m_1524 <dbl>, ep_m_2534 <dbl>, ep_m_3544 <dbl>, ep_m_4554 <dbl>, ...
```

Dünya Sağlık Örgütü tarafından toplanan bu veri seti, tüberküloz teşhisleri hakkında bilgi kaydeder. Zaten değişken olan ve yorumlanması kolay iki sütun vardır: `country` ve `year`. Bunları `sp_m_014`, `ep_m_4554` ve `rel_m_3544` gibi 56 sütun izler. Bu sütunlara yeterince uzun süre bakarsanız, bir kalıp olduğunu fark edeceksiniz. Her sütun adı _ ile ayrılmış üç parçadan oluşur. İlk parça, `sp/rel/ep`, teşhis için kullanılan yöntemi, ikinci parça, `m/f` cinsiyeti (bu veri setinde ikili bir değişken olarak kodlanmıştır) ve üçüncü parça, `014/1524/2534/3544/4554/5564/65` yaş aralığını açıklar (örneğin, 014, 0-14'ü temsil eder).

Yani bu durumda `who2`'de kaydedilmiş altı bilgi parçamız var: ülke ve yıl (zaten sütunlar); teşhis yöntemi, cinsiyet kategorisi ve yaş aralığı kategorisi (diğer sütun adlarında bulunur); ve bu kategorideki hasta sayısı (hücre değerleri). Bu altı bilgi parçasını altı ayrı sütunda düzenlemek için, `names_to` için bir sütun adı vektörü ve orijinal değişken adlarını `names_sep` için parçalara ayırmak üzere talimatlar ve `values_to` için bir sütun adı ile `pivot_longer()` kullanırız:

```
who2 |>
  pivot_longer(
```

```
    cols = !(country:year),
    names_to = c("diagnosis", "gender", "age"),
    names_sep = "_",
    values_to = "count"
  )
```

```
# A tibble: 405,440 x 6
   country      year diagnosis gender age   count
   <chr>       <dbl> <chr>     <chr>  <chr> <dbl>
 1 Afghanistan  1980 sp        m      014      NA
 2 Afghanistan  1980 sp        m      1524     NA
 3 Afghanistan  1980 sp        m      2534     NA
 4 Afghanistan  1980 sp        m      3544     NA
 5 Afghanistan  1980 sp        m      4554     NA
 6 Afghanistan  1980 sp        m      5564     NA
 7 Afghanistan  1980 sp        m      65       NA
 8 Afghanistan  1980 sp        f      014      NA
 9 Afghanistan  1980 sp        f      1524     NA
10 Afghanistan  1980 sp        f      2534     NA
# i 405,430 more rows
```

2. Pivot_wider()

Şimdiye kadar, değerlerin sütun adlarında yer aldığı yaygın sorun sınıfını çözmek için `pivot_longer()` kullandık. Şimdi, sütunları artırarak ve satırları azaltarak veri setlerini daha geniş hale getiren `pivot_wider()`'a geçeceğiz (HA HA) ve tek bir gözlemin birden çok satıra yayıldığı durumlarda yardımcı olur. Bu, gerçek hayatta daha az yaygın gibi görünüyor, ancak hükümet verileriyle uğraşırken çok ortaya çıkıyor gibi görünüyor.

Hastaların deneyimleri hakkında veri toplayan Medicare ve Medicaid Hizmetleri Merkezleri'nden bir veri seti olan `cms_patient_experience`'a bakarak başlayacağız:

```
head(cms_patient_experience)
```

```
# A tibble: 6 x 5
  org_pac_id org_nm                      measure_cd   measure_title      prf_rate
  <chr>      <chr>                       <chr>        <chr>                 <dbl>
1 0446157747 USC CARE MEDICAL GROUP INC CAHPS_GRP_1  CAHPS for MIPS SS~       63
2 0446157747 USC CARE MEDICAL GROUP INC CAHPS_GRP_2  CAHPS for MIPS SS~       87
3 0446157747 USC CARE MEDICAL GROUP INC CAHPS_GRP_3  CAHPS for MIPS SS~       86
4 0446157747 USC CARE MEDICAL GROUP INC CAHPS_GRP_5  CAHPS for MIPS SS~       57
```

```
5 0446157747 USC CARE MEDICAL GROUP INC CAHPS_GRP_8  CAHPS for MIPS SS~       85
6 0446157747 USC CARE MEDICAL GROUP INC CAHPS_GRP_12 CAHPS for MIPS SS~       24
```

İncelenen temel birim bir kuruluştur, ancak her kuruluş, anketteki her ölçüm için bir satır olmak üzere altı satıra yayılmıştır. `measure_cd` ve `measure_title` için tüm değer kümesini `distinct()` kullanarak görebiliriz:

```
cms_patient_experience |>
  distinct(measure_cd, measure_title)
```

```
# A tibble: 6 x 2
  measure_cd   measure_title
  <chr>        <chr>
1 CAHPS_GRP_1  CAHPS for MIPS SSM: Getting Timely Care, Appointments, and Infor~
2 CAHPS_GRP_2  CAHPS for MIPS SSM: How Well Providers Communicate
3 CAHPS_GRP_3  CAHPS for MIPS SSM: Patient's Rating of Provider
4 CAHPS_GRP_5  CAHPS for MIPS SSM: Health Promotion and Education
5 CAHPS_GRP_8  CAHPS for MIPS SSM: Courteous and Helpful Office Staff
6 CAHPS_GRP_12 CAHPS for MIPS SSM: Stewardship of Patient Resources
```

Bu sütunların hiçbiri özellikle harika değişken adları oluşturmaz: `measure_cd` değişkenin anlamına dair bir ipucu vermez ve `measure_title` boşluklar içeren uzun bir cümledir. Şimdilik yeni sütun adlarımızın kaynağı olarak `measure_cd`'yi kullanacağız, ancak gerçek bir analizde hem kısa hem de anlamlı olan kendi değişken adlarınızı oluşturmak isteyebilirsiniz.

`pivot_wider()` fonksiyonu, `pivot_longer()`'a zıt bir arayüze sahiptir: yeni sütun adları seçmek yerine, değerleri (`values_from`) ve sütun adını (`names_from`) tanımlayan mevcut sütunları sağlamamız gerekir:

```
cms_patient_experience |>
  pivot_wider(
    id_cols = starts_with("org"),
    names_from = measure_cd,
    values_from = prf_rate
  )
```

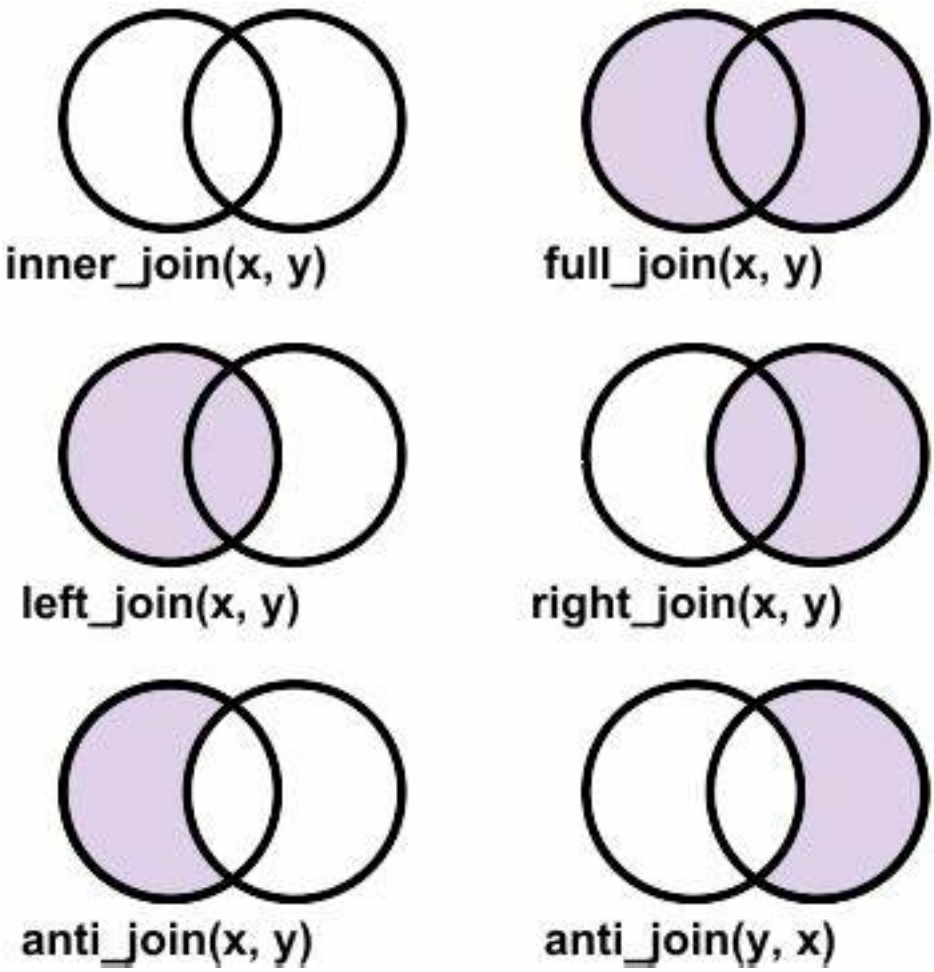
```
# A tibble: 95 x 8
   org_pac_id org_nm CAHPS_GRP_1 CAHPS_GRP_2 CAHPS_GRP_3 CAHPS_GRP_5 CAHPS_GRP_8
   <chr>      <chr>        <dbl>       <dbl>       <dbl>       <dbl>       <dbl>
 1 0446157747 USC C~          63          87          86          57          85
 2 0446162697 ASSOC~          59          85          83          63          88
```

```
 3 0547164295 BEAVE~          49          NA          75          44          73
 4 0749333730 CAPE ~          67          84          85          65          82
 5 0840104360 ALLIA~          66          87          87          64          87
 6 0840109864 REX H~          73          87          84          67          91
 7 0840513552 SCL H~          58          83          76          58          78
 8 0941545784 GRITM~          46          86          81          54          NA
 9 1052612785 COMMU~          65          84          80          58          87
10 1254237779 OUR L~          61          NA          NA          65          NA
# i 85 more rows
# i 1 more variable: CAHPS_GRP_12 <dbl>
```

İlişkisel Veri Tablolarını Birleştirme

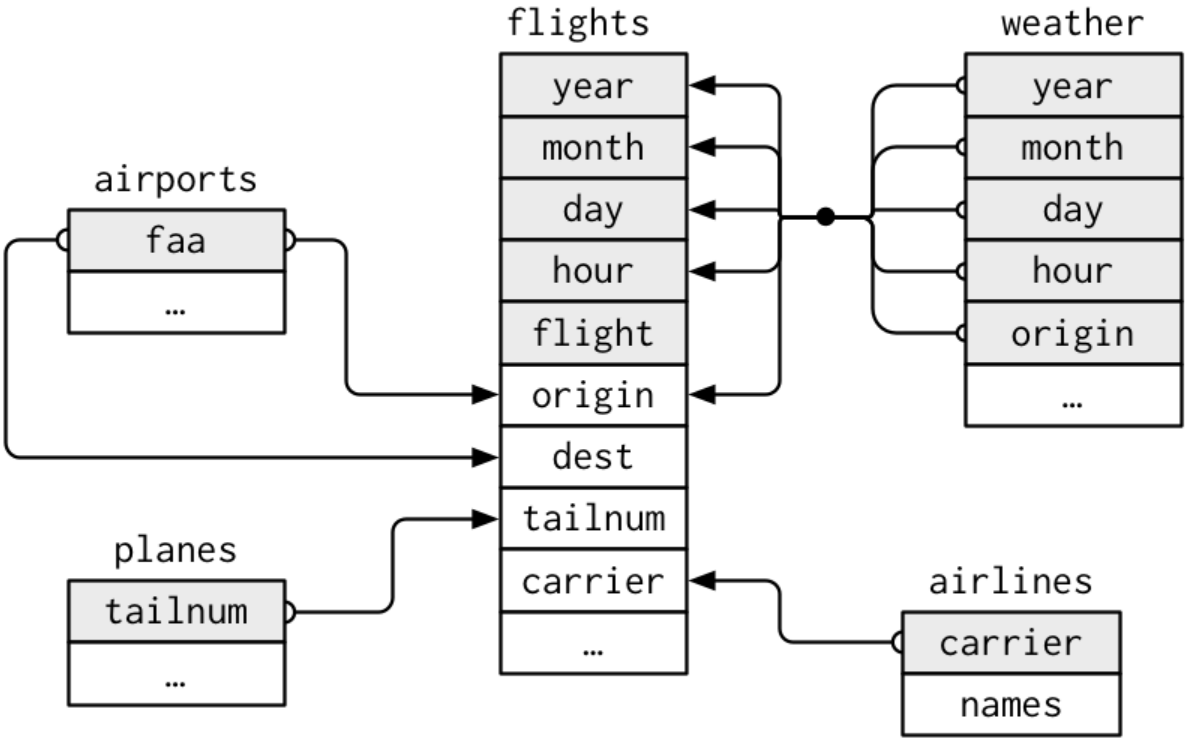
Bir veri analizinin yalnızca tek bir veri tablosu içermesi nadirdir. Genellikle birçok veri tablonuz olur ve ilgilendiğiniz soruları yanıtlamak için bunları birleştirmeniz gerekir. Birden çok veri tablosu topluca ilişkisel veri olarak adlandırılır, çünkü önemli olan yalnızca bireysel veri setleri değil, ilişkilerdir.

İlişkisel verilerle çalışmak için tablo çiftleriyle çalışan fiillere ihtiyacınız vardır:



Örnekler

`dplyr`'daki iki tablolu fiilleri kullanarak `nycflights13`'ten ilişkisel verileri inceleyeceğiz.



`nycflights13` için:

- `flights`, `planes` tablosuna `tailnum` değişkeni aracılığıyla bağlanır.
- `flights`, `airlines` tablosuna `carrier` değişkeni aracılığıyla bağlanır.
- `flights`, `airports` tablosuna iki şekilde bağlanır: `origin` ve `dest` değişkenleri aracılığıyla.
- `flights`, `weather` tablosuna `origin` (konum) ve `year`, `month`, `day` ve `hour` (zaman) değişkenleri aracılığıyla bağlanır.

`inner_join()`

`flights` tablosunun ilk birkaç satırını incelersek, `carrier` sütununun havayoluna karşılık gelen iki karakterlik bir dize içerdiğini gözlemleriz.

```
glimpse(flights)
```

```
Rows: 336,776
Columns: 19
```

```
$ year           <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2~
$ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ dep_time       <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 558, ~
$ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 600, ~
$ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2, -1~
$ arr_time       <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 849,~
$ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 851,~
$ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, -1~
$ carrier        <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6", "~
$ flight         <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301, 4~
$ tailnum        <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N394~
$ origin         <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LGA",~
$ dest           <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IAD",~
$ air_time       <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149, 1~
$ distance       <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 733, ~
$ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6~
$ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59, 0~
$ time_hour      <dttm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013-01-01 0~
```

`airlines` tablosunda, aynı iki karakterlik dizelere sahibiz, ancak aynı zamanda havayolunun tam adlarını da içeriyor.

```
head(airlines)
```

```
# A tibble: 6 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.
```

Her uçuşun bir listesini ve her bir uçuşu yöneten havayollarının tam adlarını almak için, `flights` tablosundaki satırları, her iki tabloda da `carrier` sütunu için karşılık gelen değerlere sahip olan `airlines` tablosundaki satırlarla eşleştirmemiz gerekir. Bu, `inner_join()` fonksiyonu ile gerçekleştirilir.

```r
flights_joined <- flights |>
  inner_join(airlines, by= c("carrier"="carrier"))
glimpse(flights_joined)
```

```
Rows: 336,776
Columns: 20
$ year           <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2~
$ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ dep_time       <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 558, ~
$ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 600, ~
$ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2, -1~
$ arr_time       <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 849,~
$ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 851,~
$ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, -1~
$ carrier        <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6", "~
$ flight         <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301, 4~
$ tailnum        <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N394~
$ origin         <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LGA",~
$ dest           <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IAD",~
$ air_time       <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149, 1~
$ distance       <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 733, ~
$ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6~
$ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59, 0~
$ time_hour      <dttm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013-01-01 0~
$ name           <chr> "United Air Lines Inc.", "United Air Lines Inc.", "Amer~
```

`flights_joined` veri çerçevesinin artık `name` adlı ek bir değişkene sahip olduğuna dikkat edin. Bu, artık birleştirilmiş veri çerçevesine dahil edilen `airlines` tablosundan gelen sütundur. Şifreli iki karakterlik kodlar yerine havayollarının tam adlarını görüntüleyebiliriz.

```r
flights_joined |>
  select(carrier, name, flight, origin, dest) |>
  head(3)
```

```
# A tibble: 3 x 5
  carrier name                  flight origin dest
  <chr>   <chr>                  <int> <chr>  <chr>
1 UA      United Air Lines Inc.   1545 EWR    IAH
2 UA      United Air Lines Inc.   1714 LGA    IAH
3 AA      American Airlines Inc.  1141 JFK    MIA
```

Dış Birleştirmeler (Outer Joins)

Bir iç birleştirme (`inner join`), her iki tabloda da görünen gözlemleri tutar. Bir dış birleştirme (`outer join`), tablolardan en az birinde görünen gözlemleri tutar. Üç tür dış birleştirme vardır:

- Bir sol birleştirme (`left join`), `x`'deki tüm gözlemleri tutar.
- Bir sağ birleştirme (`right join`), `y`'deki tüm gözlemleri tutar.
- Tam bir birleştirme (`full join`), `x` ve `y`'deki tüm gözlemleri tutar.

`left_join()`

`flights` ve `weather` tabloları ortak değişkenlerinde eşleşir: `year`, `month`, `day`, `hour` ve `origin`.

```
flights_joined |>
  left_join(weather)
```

```
Joining with `by = join_by(year, month, day, origin, hour, time_hour)`
```

```
# A tibble: 336,776 x 29
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# i 336,766 more rows
# i 21 more variables: 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>, name <chr>, temp <dbl>,
#   dewp <dbl>, humid <dbl>, wind_dir <dbl>, wind_speed <dbl>, wind_gust <dbl>,
#   precip <dbl>, pressure <dbl>, visib <dbl>
```

```r
flights_joined |>
  left_join(planes, by="tailnum")
```

```
# A tibble: 336,776 x 28
   year.x month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1   2013     1     1      517            515         2      830            819
 2   2013     1     1      533            529         4      850            830
 3   2013     1     1      542            540         2      923            850
 4   2013     1     1      544            545        -1     1004           1022
 5   2013     1     1      554            600        -6      812            837
 6   2013     1     1      554            558        -4      740            728
 7   2013     1     1      555            600        -5      913            854
 8   2013     1     1      557            600        -3      709            723
 9   2013     1     1      557            600        -3      838            846
10   2013     1     1      558            600        -2      753            745
# i 336,766 more rows
# i 20 more variables: 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>, name <chr>, year.y <int>,
#   type <chr>, manufacturer <chr>, model <chr>, engines <int>, seats <int>,
#   speed <int>, engine <chr>
```

Filtreleme Birleştirmeleri (Filtering Joins)

Filtreleme birleştirmeleri, gözlemleri değiştirme birleştirmeleriyle aynı şekilde eşleştirir, ancak değişkenleri değil, gözlemleri etkiler. İki tür vardır:

- `semi_join(x, y)`, y'de eşleşmesi olan x'deki tüm gözlemleri tutar.
- `anti_join(x, y)`, y'de eşleşmesi olan x'deki tüm gözlemleri bırakır.

Yarı birleştirmeler (`semi_join`), filtrelenmiş özet tablolarını orijinal satırlarla eşleştirmek için kullanışlıdır. Örneğin, en popüler on varış noktasını bulduğunuzu hayal edin:

```r
top_dest <- flights |>
  count(dest, sort=TRUE) |>
  head(10)
```

```r
flights_joined |>
  semi_join(top_dest)
```

```
Joining with `by = join_by(dest)`
```

```
# A tibble: 141,145 x 20
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      542            540         2      923            850
 2  2013     1     1      554            600        -6      812            837
 3  2013     1     1      554            558        -4      740            728
 4  2013     1     1      555            600        -5      913            854
 5  2013     1     1      557            600        -3      838            846
 6  2013     1     1      558            600        -2      753            745
 7  2013     1     1      558            600        -2      924            917
 8  2013     1     1      558            600        -2      923            937
 9  2013     1     1      559            559         0      702            706
10  2013     1     1      600            600         0      851            858
# i 141,135 more rows
# i 12 more variables: 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>, name <chr>
```

Bir yarı birleştirmenin tersi, bir karşı birleştirmedir (`anti_join`). Bir karşı birleştirme, eşleşmesi olmayan satırları tutar. Karşı birleştirmeler, birleştirme uyuşmazlıklarını teşhis etmek için kullanışlıdır. Örneğin, `flights` ve `planes` tablolarını birleştirirken, `planes` tablosunda eşleşmesi olmayan birçok uçuş olduğunu bilmek ilginizi çekebilir:

```
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 NOEGMQ    371
 8 N534MQ    364
 9 N542MQ    363
```

```
10 N531MQ    349
# i 712 more rows
```