Introduction to Data Science
Combining Different Data Sets

So far, we've only used a single dataset at a time. But frequently, we need to combine multiple datasets together for analysis, because the variables we need for analysis are spread across multiple datasets. Today, we'll walk through an example using our friend the `nycflights13` data to see how to do this.

In reality, combining different datasets together for analysis is one of the most frequently used tools when working as a data scientist. With the web, there's more data than ever before, but it's rarely all in one place, so you have to combine multiple data sources. Today's lesson will be even more useful once we learn how to import data from other sources into R in an upcoming session.

In several previous lessons, we've worked with the nycflights13 library, relying mostly on the flights data. But that library includes five different datasets:

- `Flights`, which gives us the origin and destination airport of each flight, airline code, flight time, distance, tail number (of the plane), and delay information. This is the dataset we've worked with several times now
- `Airlines`, which gives us the full name for each airline. In the flights dataset, we see only the airline code, but airline gives us the full name of each airline
- `Airports`, which gives us the exact latitude and longitude of each airport, as well as its abbreviation and time zone
- `Planes`, which gives us information about each commercial plane flying in 2013 (in the US). This includes the tail number (a unique identifier for each plane), manufacturer, year of manufacture, number of seats, and so forth
- `Weather`, which gives the weather at each of our 3 NYC airports at every hour on every day

We can combine these 5 different datasets together to answer more interesting questions than we've done so far. To use them together, we need a way of matching them up to one another. We do that by *keys*, which are variables that allow us to uniquely identify a given observation. These keys then allow us to combine the information in the two datasets.

For example, when we worked with the `flights` dataset before, I found it frustrating that the dataset only contained the carrier abbreviation, and not the full name. Some of the abbreviations are easy to remember—AA is American Airlines—but others make no sense: Jet Blue is B6, for example. To make it easier, I'd like to add the full airline name to the flights dataset. I can do this by merging (joining) flights and airlines.

But to combine these two datasets, I need a variable(s) that can link them, which is our key. What variable could serve as our key? Let's take a look the help files for each dataset to find out what they include. Let's start with flights. Remember, to pull up the help files, you need to type ?dataset, where "dataset" is the name of the dataset. When I do that for flights (with the command `?flights`), I see that it contains:

Data frame with columns

year,month,day

Date of departure

dep_time,arr_time

Actual departure and arrival times (format HHMM or HMM), local tz.

sched_dep_time,sched_arr_time

Scheduled departure and arrival times (format HHMM or HMM), local tz.

dep_delay,arr_delay

Departure and arrival delays, in minutes. Negative times represent early departures/arrivals.

hour,minute

Time of scheduled departure broken into hour and minutes.

carrier

Two letter carrier abbreviation. See airlines() to get name

tailnum

Plane tail number

flight

Flight number

origin,dest

Origin and destination. See airports() for additional metadata.

air_time

Amount of time spent in the air, in minutes

distance

Distance between airports, in miles

time_hour

```
      Scheduled date and hour of the flight as a POSIXct date. Along
      with origin, can be used to join flights data to weather data.
```

When I do the same for airlines, I find:

```
Data frame with columns

carrier

      Two letter abbreviation

name

      Full name
```

So notice that in both datasets, I have the carrier code, so I can use that as my key to match the two datasets. If I do that, I will have combined information from two different datasets to improve my analysis. Pretty neat!

Let's work through another example. Now suppose I wanted to explore how plane size relates to departure delays. I might think that larger planes take longer to board both passengers and cargo, so they're likely to be subject to longer departure delays. To test this hypothesis, I would match the flights dataset, with its information on departure delay information, with the planes dataset, which includes the number of seats on each plane (which is a measure of overall plane size).

We already have the full list of variables for flights above. Now let's look and see what we have in planes. When we do that, we find:

```
A data frame with columns:

tailnum

      Tail number

year

      Year manufactured

type

      Type of plane

manufacturer,model

      Manufacturer and model
```

`engines,seats`

`Number of engines and seats`

`speed`

`Average cruising speed in mph`

`engine`

`Type of engine`

So if I could merge these two datasets, I could see how the number of seats in each plane (using the `seats` variable in the `planes` dataset) relates to departure delays (using the `dep_delay` variable in the `flights` dataset). But how can I combine them? Remember that I need a key to match them, which is a common variable contained in both datasets. Here, notice that the variable `tailnum` (the tail number of each plane, which serves as a plane identification mechanism across carriers) is in both datasets. So I can use the tail number as the key to merge these two datasets.

This is really the first and most important step when looking to merge together two datasets: identifying a key that allows you to combine the two datasets.

When combining datasets, there are three basic types of matches you will see:

- One-to-one matches: This is when there is one observation in each dataset, and they uniquely match to one another. For example, suppose I wanted to see how county-level demographics predict election returns. I could get data on the demographic composition of each county from the U.S. Census (from https://factfinder.census.gov/faces/nav/jsf/pages/index.xhtml), and match with data on how each county voted in the 2018 election (from the Secretary of State in each state). Because each county will appear once in each dataset, it is a one-to-one match. My key would be an identifier for each county in the U.S.
- One-to-many matches: This is where there is one observation in a given dataset that matches to multiple observations in the other dataset. For example, in our example above about whether larger planes have longer departure delays, the planes dataset has each plane only appearing once, but each plane flies many flights, and we can match them up with the tailnumber key. So there is one observation in planes that matches to many observations in flights.
- Many-to-many matches: This is when there are many observations in each dataset that match to each other. These do come up from time to time, but proceed cautiously, because they are notoriously complicated. These sorts of matches are really beyond the scope of this class.

Most of the time, I find that I'm using one-to-many merges, and that's where we'll focus our attention today. You can see one-to-one matches as a special case of these one-to-many examples.

So that's the logic that underlies joining two datasets together, but now let's work on how we actually do that in R. We'll begin with our first example of wanting to add the full airline name to the flights data, which requires merging flights and airlines. Recall from the above that we have 2 datasets (`flights` and `airlines`), and a common key (`carrier`, which gives the carrier code in both). So now I'm ready to join them. Let's see this in R:

```
flights2 <- flights %>%
+   select(year:day, hour, origin, dest, tailnum, carrier)
flights2 %>%
+   left_join(airlines, by="carrier")
# A tibble: 336,776 x 9
    year month   day  hour origin dest  tailnum carrier name
   <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <chr>
 1  2013     1     1     5 EWR    IAH   N14228  UA      United Air Lines Inc.
 2  2013     1     1     5 LGA    IAH   N24211  UA      United Air Lines Inc.
 3  2013     1     1     5 JFK    MIA   N619AA  AA      American Airlines Inc.
 4  2013     1     1     5 JFK    BQN   N804JB  B6      JetBlue Airways
 5  2013     1     1     6 LGA    ATL   N668DN  DL      Delta Air Lines Inc.
 6  2013     1     1     5 EWR    ORD   N39463  UA      United Air Lines Inc.
 7  2013     1     1     6 EWR    FLL   N516JB  B6      JetBlue Airways
 8  2013     1     1     6 LGA    IAD   N829AS  EV      ExpressJet Airlines Inc.
 9  2013     1     1     6 JFK    MCO   N593JB  B6      JetBlue Airways
10  2013     1     1     6 LGA    ORD   N3ALAA  AA      American Airlines Inc.
# ... with 336,766 more rows
```

So let's walk through this code. I first create a new object flights2, which is just a smaller version of the flights data that only has the date and time information, as well as the origin, destination, tail number, and carrier information, created using the `select()` function that we discussed when we covered the dplyr library in our discussion of cleaning and transforming data. I do this so that it's easier to see the data on the screen. I then use the `left_join()` function to add in the information from the airlines dataset, and I match them up using the carrier variable, which is our key. Note that now my flights2 data has the full name of each airline, rather than just the airline code. Mission accomplished!

Different Ways of Combining Data

Let's talk through the new command `left_join()`, which is one of several ways you can combine two datasets in R. To do that, let's work through a simple example. Suppose we have the following two datasets, X and Y. I'll walk through them both with tables, and with R code.

Dataset X:

Value	Key
X1	1
X2	2
X3	3

Dataset Y:

Value	Key
Y1	1
Y2	2
Y3	4

To create these in R, we would write:

```
x <- tribble(
  ~key, ~val_x,
  1, "x1",
  2, "x2",
  3, "x3"
)
y <- tribble(
  ~key, ~val_y,
  1, "y1",
  2, "y2",
  4, "y3"
)
```

We can think about joining these two datasets in 4 different ways:

1. We can keep all observations that are common to both datasets
2. We can keep all of the observations that appear in X
3. We can keep all of the observations that appear in Y
4. We can keep all of the observations that appear in both X and Y

Let's walk through each one in turn. While we talk through them in this handout, in the video for this topic, we have animations that show them visually. This is a case where an animation is worth a thousand words, so we'd strongly encourage you to watch the video for this part, rather than just relying on the handout.

<u>Keeping all observations common to both X and Y:</u>

This is known as an `inner_join()`. Here, this would produce:

X	Y	Key
X1	Y1	1
X2	Y2	2

Note what this does: it drops cases where the key is 3 or 4, because those don't appear in both datasets. So an `inner_join()` keeps observations where the key is in both datasets, but drops observations where the key does not appear in both datasets. In R code, this is:

```
x %>%
+   inner_join(y, by="key")
# A tibble: 2 x 3
    key val_x val_y
  <dbl> <chr> <chr>
1     1 x1    y1
2     2 x2    y2
```

Note that to specify the key, we use the by option in R, and we put the variable name in quotation marks (here, that variable name is key).

<u>Keeping all of the observations that appear in X:</u>

This is a `left_join()`, which is the command we used above. Here, this would produce:

X	Y	Key
X1	Y1	1
X2	Y2	2
X3	NA	3

So here, this keeps all of the values in X. If there is a value of the key with no entry in Y, it is marked as missing data (NA). In R code:

```
> x %>%
+   left_join(y, by="key")
# A tibble: 3 x 3
    key val_x val_y
  <dbl> <chr> <chr>
1     1 x1    y1
2     2 x2    y2
3     3 x3    NA
```

Writing the code this way specifies why we call it a "left" join: our X dataset is to the left of the pipe.

<u>Keeping all of the observations that appear in Y:</u>

This is a `right_join()`, which is the converse of the command above. Here, this would produce:

X	Y	Key
X1	Y1	1
X2	Y2	2
NA	Y3	4

So here, this keeps all of the values in Y, so it's the flip side of the `left_join()` command. In R code:

```
x %>%
+   right_join(y, by="key")
# A tibble: 3 x 3
    key val_x val_y
  <dbl> <chr> <chr>
1     1 x1    y1
2     2 x2    y2
3     4 NA    y3
```

<u>Keeping all observations in either X or Y:</u>

This is known as a `full_join()`:

X	Y	Key
X1	Y1	1
X2	Y2	2
X3	NA	3
NA	Y3	4

Note here that this includes all observations, even if they only appear in one dataset.

```
x %>%
+   full_join(y, by="key")
# A tibble: 4 x 3
    key val_x val_y
  <dbl> <chr> <chr>
1     1 x1    y1
2     2 x2    y2
3     3 x3    NA
4     4 NA    y3
```

So now let's return to our example above, where we added the airline names into the flights dataset. We used the `left_join()` command because we started with the flights data (which is our X data), and we added in the airline names from the airline dataset (which is our Y data).

We kept all of the data in the flights dataset (X), and added the airline name from airlines (Y). We matched the two datasets using the carrier variable as our key.

99% of the time, if you're going to do this, you're going to use the `left_join()` option, because you typically have a dataset (like flights), and you want to use another dataset to augment it with additional information.

TEST YOURSELF: Earlier, we said we wanted to see if larger planes were more subject to departure delays. To do this, let's break it down into steps where we merge the datasets to have all the necessary data in one dataframe, and then we can use that newly merged dataset to do more interesting analysis.

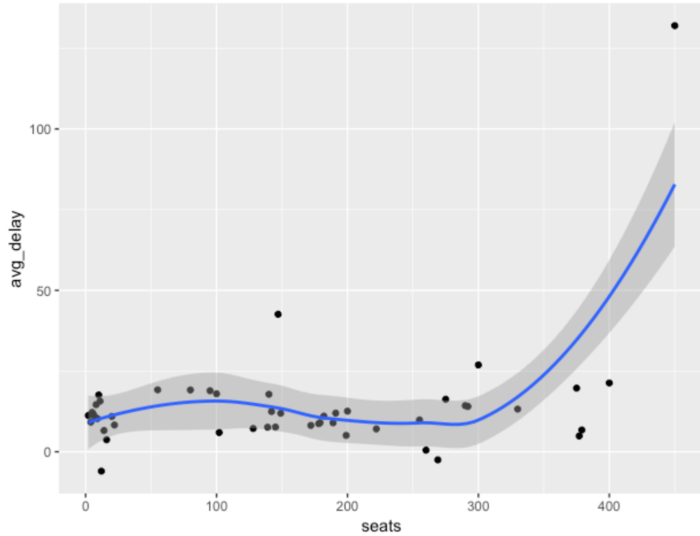
a. Merge the flights and planes datasets. Think about what type of join is necessary and what key is shared to combine the two.
b. Once you have this new dataset, use the number of seats as a measure of plane size and plot the relationship between size and average departure delay.

ANSWER:

```
## Test Yourself
flights %>%
  left_join(planes, by="tailnum") %>%
  filter(!is.na(dep_delay)) %>%
  group_by(seats) %>%
  summarize(avg_delay = mean(dep_delay)) %>%
  ggplot(mapping=aes(x=seats,y=avg_delay)) +
  geom_point() +
  geom_smooth()
```

Note what I've done here:

1. I use `left_join()` to join the two datasets by the plane tail number (remember, we showed above that this is the key that allows us to join these datasets).
2. I removed observations where departure delay is missing, which are flights that are cancelled (since they're not relevant to the relationship we're examining).
3. Then I group observations by the number of seats and find the average delay per plane size.
4. Finally, I plot the results using `ggplot`.



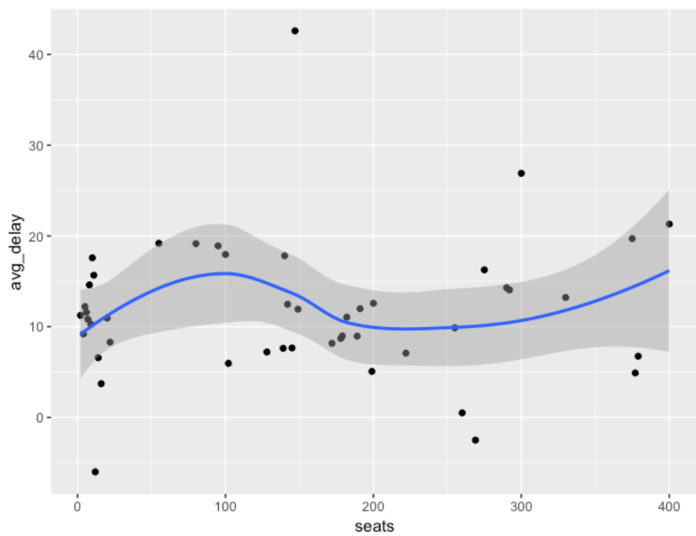
Hmm, it looks like the pattern is really being driven by planes greater than 400 seats. I wonder how many of those are in our dataset?

```
> arrange(planes,desc(seats))
# A tibble: 3,322 x 9
   tailnum  year type                    manufacturer model   engines seats speed engine
   <chr>   <int> <chr>                   <chr>        <chr>     <int> <int> <int> <chr>
 1 N670US   1990 Fixed wing multi engine BOEING       747-451       4   450    NA Turbo-jet
 2 N206UA   1999 Fixed wing multi engine BOEING       777-222       2   400    NA Turbo-fan
 3 N228UA   2002 Fixed wing multi engine BOEING       777-222       2   400    NA Turbo-fan
 4 N272AT     NA Fixed wing multi engine BOEING       777-200       2   400    NA Turbo-jet
 5 N57016   2000 Fixed wing multi engine BOEING       777-224       2   400    NA Turbo-fan
 6 N77012   1999 Fixed wing multi engine BOEING       777-224       2   400    NA Turbo-fan
 7 N777UA   1995 Fixed wing multi engine BOEING       777-222       2   400    NA Turbo-fan
 8 N78003   1998 Fixed wing multi engine BOEING       777-224       2   400    NA Turbo-fan
 9 N78013   1999 Fixed wing multi engine BOEING       777-224       2   400    NA Turbo-fan
10 N787UA   1997 Fixed wing multi engine BOEING       777-222       2   400    NA Turbo-fan
# ... with 3,312 more rows
```

There's only one plane, a legacy 747. So this is likely just an idiosyncrasy related to that plane, so let's drop it and focus on planes with 400 or fewer seats:

```
flights %>%
  left_join(planes, by="tailnum") %>%
  filter(!is.na(dep_delay) & seats < 401) %>%
  group_by(seats) %>%
  summarize(avg_delay = mean(dep_delay)) %>%
  ggplot(mapping=aes(x=seats,y=avg_delay)) +
  geom_point() +
  geom_smooth()
```

Which gives us the following graph:



So here, contrary to our expectations, we don't really see a pattern. If anything, small-ish planes (those around 100 seats) are the ones who are most likely to be delayed. If we wanted to explore this pattern further, we'd have to dig into aspects of those flights. But the key thing here is to understand the use of `left_join()` and to get some more practice with our other key commands.

Being Careful with Keys

It's worth commenting on the use of keys to match datasets. Above, we showed how you use the `by` option to specify the key. But what happens if you omit it? Then R will match all variables with common names across the two datasets; R calls this a "natural join". For example, suppose I wanted to merge the weather data into the flights data:

```
>flights2 %>%
+   left_join(weather)
```

This gives us the following output in R:

```
Joining, by = c("year", "month", "day", "hour", "origin")
# A tibble: 336,776 x 18
    year month   day  hour origin dest  tailnum carrier  temp  dewp humid wind_dir wind_speed wind_gust
   <dbl> <dbl> <int> <dbl> <chr>  <chr> <chr>   <chr>   <dbl> <dbl> <dbl>    <dbl>      <dbl>     <dbl>
 1  2013     1     1     5 EWR    IAH   N14228  UA       39.0  28.0  64.4      260       12.7      NA  
 2  2013     1     1     5 LGA    IAH   N24211  UA       39.9  25.0  54.8      250       15.0      21.9
 3  2013     1     1     5 JFK    MIA   N619AA  AA       39.0  27.0  61.6      260       15.0      NA  
 4  2013     1     1     5 JFK    BQN   N804JB  B6       39.0  27.0  61.6      260       15.0      NA  
 5  2013     1     1     6 LGA    ATL   N668DN  DL       39.9  25.0  54.8      260       16.1      23.0
 6  2013     1     1     5 EWR    ORD   N39463  UA       39.0  28.0  64.4      260       12.7      NA  
 7  2013     1     1     6 EWR    FLL   N516JB  B6       37.9  28.0  67.2      240       11.5      NA  
 8  2013     1     1     6 LGA    IAD   N829AS  EV       39.9  25.0  54.8      260       16.1      23.0
 9  2013     1     1     6 JFK    MCO   N593JB  B6       37.9  27.0  64.3      260       13.8      NA  
10  2013     1     1     6 LGA    ORD   N3ALAA  AA       39.9  25.0  54.8      260       16.1      23.0
# ... with 336,766 more rows, and 4 more variables: precip <dbl>, pressure <dbl>, visib <dbl>,
#   time_hour <dttm>
```

So here, R tells us that it's joining by all of the common variables in both datasets: Year, month, day, hour, and origin.

So why not always do this? You can run into problems if a variable in one dataset does not correspond to a variable in another dataset. Let's take the datasets we merged above in our "Test Yourself" problem: `flights` and `planes`. In the `flights` dataset, the year variable corresponds to the year of the flight, which is always 2013. But in the `planes` dataset, the year variable corresponds to the year in which the plane was manufactured (see the help files for both datasets above). So if I naively try the natural join `flights` and `planes`, R will assume that the year variable in both should be matched, and I'll get problematic output:

```
> flights2 %>%
+   left_join(planes)
Joining, by = c("year", "tailnum")
# A tibble: 336,776 x 15
    year month   day  hour origin dest  tailnum carrier type  manufacturer model engines seats speed
engine
   <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <chr> <chr>        <chr>   <int> <int> <int>
<chr>
 1  2013     1     1     5 EWR    IAH   N14228  UA      NA    NA           NA         NA    NA    NA NA
 2  2013     1     1     5 LGA    IAH   N24211  UA      NA    NA           NA         NA    NA    NA NA
 3  2013     1     1     5 JFK    MIA   N619AA  AA      NA    NA           NA         NA    NA    NA NA
 4  2013     1     1     5 JFK    BQN   N804JB  B6      NA    NA           NA         NA    NA    NA NA
 5  2013     1     1     6 LGA    ATL   N668DN  DL      NA    NA           NA         NA    NA    NA NA
 6  2013     1     1     5 EWR    ORD   N39463  UA      NA    NA           NA         NA    NA    NA NA
 7  2013     1     1     6 EWR    FLL   N516JB  B6      NA    NA           NA         NA    NA    NA NA
 8  2013     1     1     6 LGA    IAD   N829AS  EV      NA    NA           NA         NA    NA    NA NA
 9  2013     1     1     6 JFK    MCO   N593JB  B6      NA    NA           NA         NA    NA    NA NA
10  2013     1     1     6 LGA    ORD   N3ALAA  AA      NA    NA           NA         NA    NA    NA NA
# ... with 336,766 more rows
```

So note that didn't work: I get NAs for all values of every observation from planes! The reason why is that no planes were manufactured in 2013, so R didn't know how to match on this variable. Instead, I need to explicitly tell R how to match using the by option:

```
> flights2 %>%
+   left_join(planes, by="tailnum")
# A tibble: 336,776 x 16
   year.x month   day  hour origin dest  tailnum carrier year.y type  manufacturer model engines seats
    <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>    <int> <chr> <chr>        <chr>   <int> <int>
 1   2013     1     1     5 EWR    IAH   N14228  UA        1999 Fixe… BOEING       737-…       2   149
 2   2013     1     1     5 LGA    IAH   N24211  UA        1998 Fixe… BOEING       737-…       2   149
 3   2013     1     1     5 JFK    MIA   N619AA  AA        1990 Fixe… BOEING       757-…       2   178
 4   2013     1     1     5 JFK    BQN   N804JB  B6        2012 Fixe… AIRBUS       A320…       2   200
 5   2013     1     1     6 LGA    ATL   N668DN  DL        1991 Fixe… BOEING       757-…       2   178
 6   2013     1     1     5 EWR    ORD   N39463  UA        2012 Fixe… BOEING       737-…       2   191
 7   2013     1     1     6 EWR    FLL   N516JB  B6        2000 Fixe… AIRBUS INDU… A320…       2   200
 8   2013     1     1     6 LGA    IAD   N829AS  EV        1998 Fixe… CANADAIR     CL-6…       2    55
 9   2013     1     1     6 JFK    MCO   N593JB  B6        2004 Fixe… AIRBUS       A320…       2   200
10   2013     1     1     6 LGA    ORD   N3ALAA  AA          NA NA    NA           NA         NA    NA
# ... with 336,766 more rows, and 2 more variables: speed <int>, engine <chr>
```

Now I get the correct information from join. There's an important lesson here: make sure you know what your variables mean!

The key can also have different names across datasets. So far, we've had the key have the same name in both datasets, which is good practice. But it is not strictly speaking necessary. For example, I could add the latitude/longitude data for each destination airport into the flights data using the airports dataset:

```
> flights2 %>%
+   left_join(airports, c("dest" = "faa"))
# A tibble: 336,776 x 15
    year month   day  hour origin dest  tailnum carrier name            lat   lon   alt    tz dst   tzone
   <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <chr>         <dbl> <dbl> <int> <dbl> <chr> <chr>
```

```
 1  2013     1     1      5 EWR    IAH   N14228  UA      George Bush… 30.0 -95.3    97    -6 A     Americ…
 2  2013     1     1      5 LGA    IAH   N24211  UA      George Bush… 30.0 -95.3    97    -6 A     Americ…
 3  2013     1     1      5 JFK    MIA   N619AA  AA      Miami Intl   25.8 -80.3     8    -5 A     Americ…
 4  2013     1     1      5 JFK    BQN   N804JB  B6      NA           NA    NA      NA    NA NA    NA
 5  2013     1     1      6 LGA    ATL   N668DN  DL      Hartsfield … 33.6 -84.4  1026    -5 A     Americ…
 6  2013     1     1      5 EWR    ORD   N39463  UA      Chicago Oha… 42.0 -87.9   668    -6 A     Americ…
 7  2013     1     1      6 EWR    FLL   N516JB  B6      Fort Lauder… 26.1 -80.2     9    -5 A     Americ…
 8  2013     1     1      6 LGA    IAD   N829AS  EV      Washington … 38.9 -77.5   313    -5 A     Americ…
 9  2013     1     1      6 JFK    MCO   N593JB  B6      Orlando Intl 28.4 -81.3    96    -5 A     Americ…
10  2013     1     1      6 LGA    ORD   N3ALAA  AA      Chicago Oha… 42.0 -87.9   668    -6 A     Americ…
# ... with 336,766 more rows
```

Note that even though the airport code for the destination was stored as `faa` in the `airports` dataset, and as `dest` in the `flights` dataset, R can match them up using this syntax. In the syntax above, the variable name in the X dataset (`flights`) goes first, and the variable name in the Y dataset (`airports`) goes second.

TEST YOURSELF: Inspired by the problem above, how would you add the latitude and longitude for both the origin and the destination airports to the dataset we just produced? You might want to do this to draw a map, for example.

ANSWER:

```
> flights2 %>%
+   left_join(airports, c("dest" = "faa")
+             ) %>%
+   left_join(airports, c("origin" = "faa"))
# A tibble: 336,776 x 22
    year month   day  hour origin dest  tailnum carrier name.x lat.x lon.x alt.x  tz.x dst.x tzone.x name.y
   <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <chr>  <dbl> <dbl> <int> <dbl> <chr> <chr>   <chr>
 1  2013     1     1      5 EWR    IAH   N14228  UA      Georg…  30.0 -95.3    97    -6 A     Americ… Newar…
 2  2013     1     1      5 LGA    IAH   N24211  UA      Georg…  30.0 -95.3    97    -6 A     Americ… La Gu…
 3  2013     1     1      5 JFK    MIA   N619AA  AA      Miami…  25.8 -80.3     8    -5 A     Americ… John …
 4  2013     1     1      5 JFK    BQN   N804JB  B6      NA      NA    NA      NA    NA NA    NA      John …
 5  2013     1     1      6 LGA    ATL   N668DN  DL      Harts…  33.6 -84.4  1026    -5 A     Americ… La Gu…
 6  2013     1     1      5 EWR    ORD   N39463  UA      Chica…  42.0 -87.9   668    -6 A     Americ… Newar…
 7  2013     1     1      6 EWR    FLL   N516JB  B6      Fort …  26.1 -80.2     9    -5 A     Americ… Newar…
 8  2013     1     1      6 LGA    IAD   N829AS  EV      Washi…  38.9 -77.5   313    -5 A     Americ… La Gu…
 9  2013     1     1      6 JFK    MCO   N593JB  B6      Orlan…  28.4 -81.3    96    -5 A     Americ… John …
10  2013     1     1      6 LGA    ORD   N3ALAA  AA      Chica…  42.0 -87.9   668    -6 A     Americ… La Gu…
# ... with 336,766 more rows, and 6 more variables: lat.y <dbl>, lon.y <dbl>, alt.y <int>, tz.y <dbl>,
#   dst.y <chr>, tzone.y <chr>
```

Note what happens! Because each airport has a set of associated variables (latitude, longitude, altitude, time zone, and so forth), there are 2 sets of these variables now: one for the destination airport and one for the originating airport. To avoid confusion, R appends the suffix `.x` to the first set of variables (here, the destination airport), and the suffix `.y` to the second set (here, the originating airport). To tell which one is x and which one is y, just look at the R code chunk above: we first merged in the destination airport, so it is x, and then we merged in the origin airport, so it is y.

In these cases, it's often easier to clarify by using the suffix argument:

```
 > flights2 %>%
+   left_join(airports, c("dest" = "faa")
+   ) %>%
+   left_join(airports, c("origin" = "faa"),
+             suffix=c("_dest","_origin"))
# A tibble: 336,776 x 22
    year month   day  hour origin dest  tailnum carrier name_dest lat_dest lon_dest alt_dest tz_dest
   <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <chr>        <dbl>    <dbl>    <int>   <dbl>
 1  2013     1     1      5 EWR    IAH   N14228  UA      George B…     30.0    -95.3       97      -6
 2  2013     1     1      5 LGA    IAH   N24211  UA      George B…     30.0    -95.3       97      -6
 3  2013     1     1      5 JFK    MIA   N619AA  AA      Miami In…     25.8    -80.3        8      -5
 4  2013     1     1      5 JFK    BQN   N804JB  B6      NA            NA       NA         NA      NA
```

```
 5  2013     1     1      6 LGA    ATL    N668DN  DL      Hartsfie…     33.6    -84.4     1026      -5
 6  2013     1     1      5 EWR    ORD    N39463  UA      Chicago …     42.0    -87.9      668      -6
 7  2013     1     1      6 EWR    FLL    N516JB  B6      Fort Lau…     26.1    -80.2        9      -5
 8  2013     1     1      6 LGA    IAD    N829AS  EV      Washingt…     38.9    -77.5      313      -5
 9  2013     1     1      6 JFK    MCO    N593JB  B6      Orlando …     28.4    -81.3       96      -5
10  2013     1     1      6 LGA    ORD    N3ALAA  AA      Chicago …     42.0    -87.9      668      -6
# ... with 336,766 more rows, and 9 more variables: dst_dest <chr>, tzone_dest <chr>, name_origin <chr>,
#   lat_origin <dbl>, lon_origin <dbl>, alt_origin <int>, tz_origin <dbl>, dst_origin <chr>,
#   tzone_origin <chr>
```

Note what this does: it appends the suffix `_dest` to the variables describing the destination airport, and appends the suffix `_origin` to the variables describing the originating airport. This seems silly, but having clear variable labels will save you many, many headaches later on when conducting analysis!

Joining with Filtered Data

So far, we've covered the most common types of joins. But there is another class of joins that are useful when filtering observations. The way this typically comes up is when you have a filtered summary table that you want to then match back to the data. For example, suppose I wanted to find the 5 most popular destinations from New York:

```
top5 <- flights %>%
+   count(dest, sort=T) %>%
+   head(5)
> top5
# A tibble: 5 x 2
  dest      n
  <chr> <int>
1 ORD   17283
2 ATL   17215
3 LAX   16174
4 BOS   15508
5 MCO   14082
```

So here, the 5 most popular destinations are O'Hare (Chicago), Atlanta, Los Angeles, Boston, and Orlando. You know what `count()` does, and remember that `head(X)` just gives us the first X rows (so here, the 5 airports with the most flights from New York).

Now suppose I wanted to find all flights that went to one of those destinations. I'd use a function called `semi_join()`:

```
flights %>%
+   semi_join(top5)
Joining, by = "dest"
# A tibble: 80,262 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl> <chr>    <int>
 1  2013     1     1      554            600        -6      812            837       -25 DL         461
 2  2013     1     1      554            558        -4      740            728        12 UA        1696
 3  2013     1     1      557            600        -3      838            846        -8 B6          79
 4  2013     1     1      558            600        -2      753            745         8 AA         301
 5  2013     1     1      558            600        -2      924            917         7 UA         194
 6  2013     1     1      559            559         0      702            706        -4 B6        1806
 7  2013     1     1      600            600         0      837            825        12 MQ        4650
 8  2013     1     1      606            610        -4      837            845        -8 DL        1743
 9  2013     1     1      608            600         8      807            735        32 MQ        3768
```

```
10  2013     1     1      615            615         0      833            842        -9 DL         575
# ... with 80,252 more rows, and 8 more variables: tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

And now we have the 80,262 flights from New York to one of these destinations during 2013. Note that if you sum up the flights to those 5 destinations, you'll see that those 5 destinations have 17,283 + 17,215 + 16,174 + 15,508 + 14,082 = 80,262 flights.

Formally, a semi-join keeps all observations of X that have a match in Y. So let's go back to our earlier X and Y:

Dataset X:

Value	Key
X1	1
X2	2
X3	3

Dataset Y:

Value	Key
Y1	1
Y2	2
Y3	4

So now if we use semi-join on x and y, we find:

```
x %>%
+   semi_join(y)
Joining, by = "key"
# A tibble: 2 x 2
    key val_x
  <dbl> <chr>
1     1 x1
2     2 x2
```

R gives us every value of X where there's a corresponding value of the key in Y. So here, because the key values 1 and 2 appear in Y, it returns X1 and X2. In the same way, in our example above, it selected out only flights to one of our top 5 destinations (ORD, ATL, LAX, BOS, and MCO).

What's the difference between an `inner_join()` and a `semi_join()`? Formally, R's help file tells us that "A semi join differs from an inner join because an inner join will return one row of x for each matching row of y, where a semi join will never duplicate rows of x." I personally find that completely unhelpful, but more power to you if that's clear. To me, the difference is that a `semi_join()` is what you use when you've filtered down your data, as we did above. With `top5`, we filtered the full dataset of 336,766 flights to contain only the 80,252 flights to the five most common destinations. Because we had this filtered data, I used a `semi_join()`.

There's also the opposite of this, called an `anti_join()`, which keeps the rows that don't have a match. So continuing our X and Y example:

```
x %>%
+   anti_join(y)
Joining, by = "key"
# A tibble: 1 x 2
    key val_x
  <dbl> <chr>
1     3 x3
```

So here, because the key values 1 and 2 appear in Y, it drops those, and reports on the key value (3) in X that does not appear in Y.

This is useful mostly for diagnosing errors in your data. For example, let's anti-join the `planes` data to the `flights` data (using the tail number of each aircraft). Remember, this will tell us what observations in `flights` do not appear in the `planes` dataset. This is telling us when we don't have information on the plane that flew a given flight. Let's do this, and look at the result:

```
flights %>%
+   anti_join(planes, by="tailnum") %>%
+   count(tailnum, sort=T)
# A tibble: 722 x 2
   tailnum     n
   <chr>   <int>
 1 NA       2512
 2 N725MQ    575
 3 N722MQ    513
 4 N723MQ    507
 5 N713MQ    483
 6 N735MQ    396
 7 N0EGMQ    371
 8 N534MQ    364
 9 N542MQ    363
10 N531MQ    349
# ... with 712 more rows
```

Why do 2,512 flights have a missing value of the tail number? What does that mean? Let's take a closer look:

```
> flights %>%
```

```
+   anti_join(planes, by="tailnum") %>%
+   filter(., is.na(tailnum)) %>%
+   count(carrier)
# A tibble: 7 x 2
  carrier     n
  <chr>   <int>
1 9E       1044
2 AA         84
3 F9          3
4 MQ          2
5 UA        686
6 US        663
7 WN         30
```

Wow, almost ½ of the missing observations are from 9E (Endeavor Air). It turns out that they don't use a tail number to identify their planes, so this means that we'd need another way to identify their planes if we wanted to learn more about them. So here, using `anti_join()` showed us an important limitation/caveat in our data.

TEST YOURSELF: Write the R code you would use to find all flights flown by planes that flew at least 50 flights in 2013.

ANSWER:
```
flew50 <- flights %>%
  group_by(tailnum) %>%
  count() %>%
  filter(n > 49)
flights %>%
  semi_join(flew50)
```