Data Transformation and Exploration

Lecture 03.2: Relational Data

Lauren Sullivan

Module: Data Management, Visualization & Reproducibility

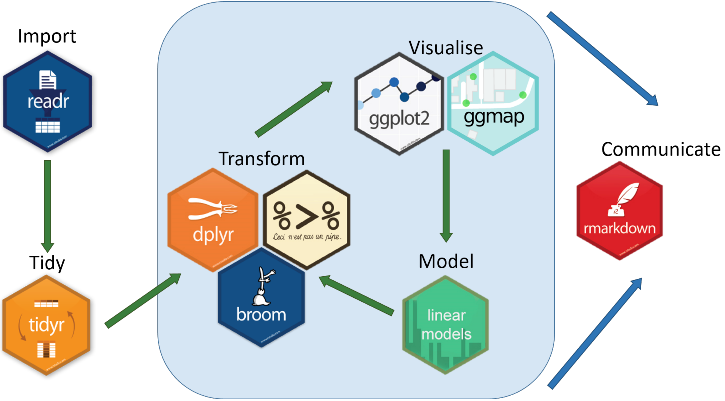
Relational Data

As we mentioned previously, it is good practice to maintain smaller datasets and then merge them together through code. Using multiple tables of data is called *Relational Data* because we are interested in the relations between datasets, not individual ones.

Relations are always built between pairs of tables. And to do this work we need some terminology.

- **Mutating joins**: adds a new variable to one data frame from matching observations in another
- **Filtering joins**: filters observations from one data frame based on if they match an observation from another data frame.
- **Set operations**: treats observations as if they were set elements.

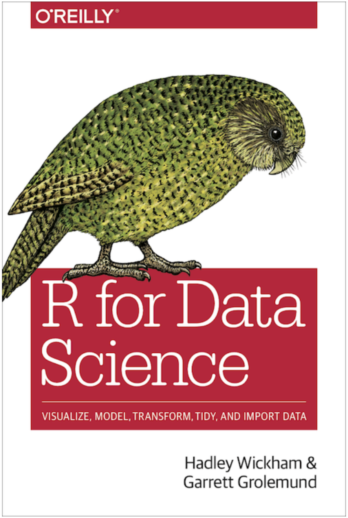
Relational Data with dplyr



In this lecture we will use the `library(nycflights13)` from R for Data Science. This library contains data on flights in and out of NYC.

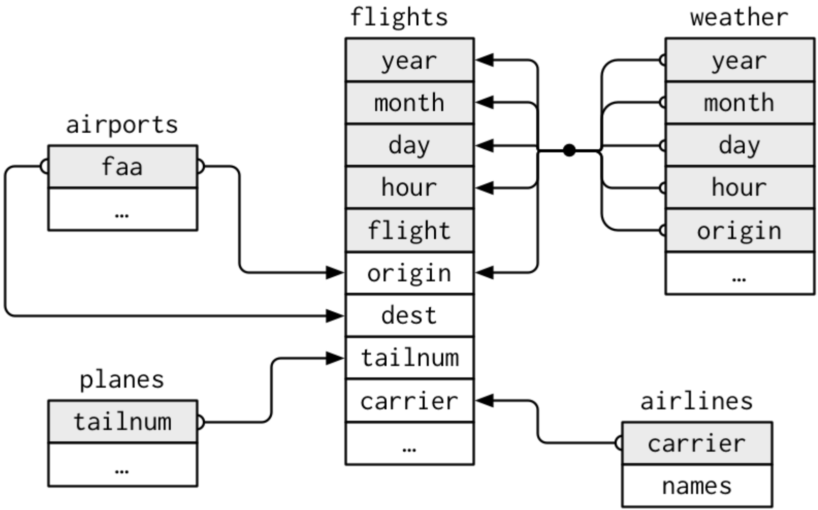
```
library(nycflights13)

airlines  # describes all airlines in NYC

airports  # describes the airports flights go to/from

planes    # describes each plane

weather   # weather at each NYC airport each hour
```

For More Information

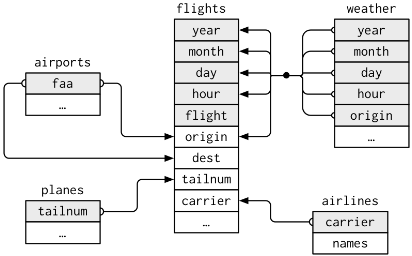


R for Data Science.

Relational data for nycflights13



Relational data for nycflights13



- `flights` connects to `planes` via a single variable, `tailnum`.
- `flights` connects to `airlines` through the `carrier` variable.
- `flights` connects to `airports` in two ways: via the `origin` and `dest` variables.
- `flights` connects to `weather` via `origin` (the location), and `year`, `month`, `day` and `hour` (the time).

Keys

The terminology used by *R for Data Science* for a variable (or set of variables) that connects each pair of tables.

- There are three types of keys

    1. *Primary Key* - uniquely identifies an observation in its own table. Here, `planes$tailnum` uniqely identifies each plane in the `planes` table.
    2. *Foreign Key* - uniquely identifies an observation in another table. Here, `flights$tailnum` appears in the `flights` table where it matches each flight to a unique plane in `planes`.
    3. *Surrogate Key* - sometimes datasets do not contain a primary key, so you must make one. Here, `flights$tailnum` is not a primary key because the same plane appears multiple times per day at the airport. Create one using, for example, `mutate()` and `row_number()`. **This then becomes a *primary key.***

A *primary key* that corresponds to a *foreign key* is a relation. This builds the 1:1, 1:n, or n:1 relationships we mentioned previously.

Mutating Joins

Mutating joins combine variables from two tables.

Let's make `flights` smaller so we can better see our data.

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

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

Mutating Joins

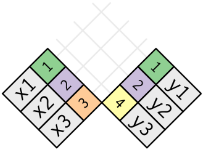
To add the full name of the airline to `flights2`, you combine `airlines` with `flights2` with `left_join()`.

```
flights2 %>%
  select(-origin, -dest) %>%
  left_join(airlines, by = "carrier")
```

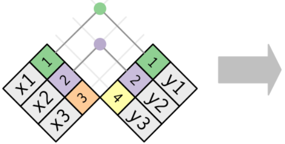
```
## # A tibble: 336,776 x 7
##     year month   day  hour tailnum carrier name
##    <int> <int> <int> <dbl> <chr>   <chr>   <chr>
##  1  2013     1     1     5 N14228  UA      United Air Lines Inc.
##  2  2013     1     1     5 N24211  UA      United Air Lines Inc.
##  3  2013     1     1     5 N619AA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

Joins, How Do They Work?

Imagine you have two datasets, x (on the left) and y (on the right), and they each have *key* variables (colored column). The following diagram shows all potential matches with the grid lines



However the only possible matches are indicated with colored dots. The resulting dataset demonstrates that in this case all variables are joined, and some observations are lost.

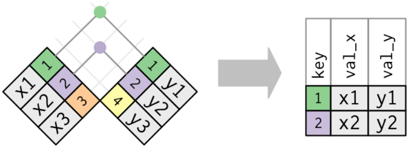


key	val_x	val_y
1	x1	y1
2	x2	y2

Inner Joins

Inner joins match pairs of observations based on equal key variables, but they lose observations without matches.

- You specify the key variable with `by`.




```
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
```

Outer Joins

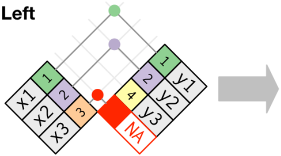
Outer joins also match observations based on key variables, but **do not** lose observations.

- There are three types of outer joins.
  1. A **left join** keeps all observations in `x`.
  2. A **right join** keeps all observations in `y`.
  3. A **full join** keeps all observations in `x` and `y`.

In all cases, unmatched observations are filled with `NA`.

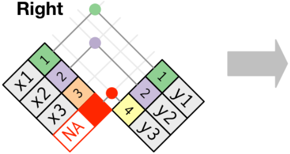
Outer Joins

Left



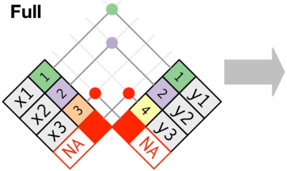
key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

Right



key	val_x	val_y
1	x1	y1
2	x2	y2
4	NA	y3

Full

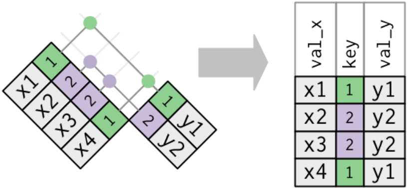


key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

Duplicate Keys

Sometimes keys are not unique. There are two possibilities here.

1. One table has duplicate keys (a 1:n relationship)

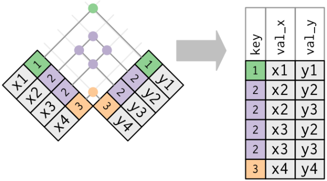



```
left_join(x, y, by = "key")
```

```
## # A tibble: 4 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     2 x3    y2
## 4     1 x4    y1
```

Duplicate Keys

2. Both tables have duplicate keys. This is often an error because neither table has a unique identifier.




```
left_join(x, y, by = "key")
```

```
## # A tibble: 6 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     2 x2    y3
## 4     2 x3    y2
## 5     2 x3    y3
## 6     3 x4    y4
```

Defining the Key Columns

Natural joins, or `by = NULL` uses all variables that are in both tables. The `flights` and `weather` table have `year`, `month`, `day`, `hour`, and `origin` in common.

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

```
## 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
##    <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <dbl> <dbl> <dbl>
##  1  2013     1     1     5 EWR    IAH   N14228  UA       39.0  28.0  64.4
##  2  2013     1     1     5 LGA    IAH   N24211  UA       39.9  25.0  54.8
##  3  2013     1     1     5 JFK    MIA   N619AA  AA       39.0  27.0  61.6
##  4  2013     1     1     5 JFK    BQN   N804JB  B6       39.0  27.0  61.6
##  5  2013     1     1     6 LGA    ATL   N668DN  DL       39.9  25.0  54.8
##  6  2013     1     1     5 EWR    ORD   N39463  UA       39.0  28.0  64.4
##  7  2013     1     1     6 EWR    FLL   N516JB  B6       37.9  28.0  67.2
##  8  2013     1     1     6 LGA    IAD   N829AS  EV       39.9  25.0  54.8
##  9  2013     1     1     6 JFK    MCO   N593JB  B6       37.9  27.0  64.3
## 10  2013     1     1     6 LGA    ORD   N3ALAA  AA       39.9  25.0  54.8
## # ... with 336,766 more rows, and 7 more variables: wind_dir <dbl>,
## #   wind_speed <dbl>, wind_gust <dbl>, precip <dbl>, pressure <dbl>,
## #   visib <dbl>, time_hour <dttm>
```

Defining the Key Columns

You can join by a *character vector* with `by = "x"`. Try joining `flights` and `planes` by `tailnum`.

```
flights2 %>%
  left_join(planes, by = "tailnum")
```

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

You can see that both datasets had a `year` column but they had different values, so the resulting variables are `year.x` and `year.y`.

Defining the Key Columns

A *named character vector*, for example `by = c("a" = "b")` allows you to match variable `a` in table `x` to variable `b` in table `y`. Note: the variables from `x` will always be used in the output.

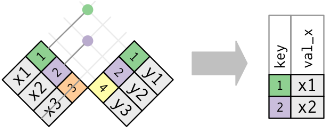
```
flights2 %>%
  left_join(airports, c("dest" = "faa"))
```

```
## # A tibble: 336,776 x 15
##     year month   day  hour origin dest  tailnum carrier name     lat   lon   alt
##    <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <chr>  <dbl> <dbl> <dbl>
##  1  2013     1     1     5 EWR    IAH   N14228  UA      Georg~  30.0 -95.3    97
##  2  2013     1     1     5 LGA    IAH   N24211  UA      Georg~  30.0 -95.3    97
##  3  2013     1     1     5 JFK    MIA   N619AA  AA      Miami~  25.8 -80.3     8
##  4  2013     1     1     5 JFK    BQN   N804JB  B6      <NA>    NA    NA      NA
##  5  2013     1     1     6 LGA    ATL   N668DN  DL      Harts~  33.6 -84.4  1026
##  6  2013     1     1     5 EWR    ORD   N39463  UA      Chica~  42.0 -87.9   668
##  7  2013     1     1     6 EWR    FLL   N516JB  B6      Fort ~  26.1 -80.2     9
##  8  2013     1     1     6 LGA    IAD   N829AS  EV      Washi~  38.9 -77.5   313
##  9  2013     1     1     6 JFK    MCO   N593JB  B6      Orlan~  28.4 -81.3    96
## 10  2013     1     1     6 LGA    ORD   N3ALAA  AA      Chica~  42.0 -87.9   668
## # ... with 336,766 more rows, and 3 more variables: tz <dbl>, dst <chr>,
## #   tzone <chr>
```

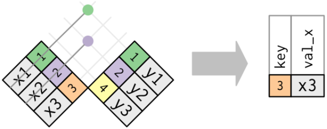
Filtering Joins

Filtering joins only affect observations, not variables.

- `semi_join(x, y)` **keeps** observations in `x` with a match in `y`.



- `anti_join(x, y)` **drops** observations in `x` with a match in `y`.



semi_join()

So say you have a table of the top destinations (`top_dest`) and you want to know what flights go to those destinations.

```
trips <- flights %>%
          semi_join(top_dest)
```

```
## Joining, by = "dest"
```

```
trips[,1:7]
```

```
## # A tibble: 141,145 x 7
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      542            540         2      923
##  2  2013     1     1      554            600        -6      812
##  3  2013     1     1      554            558        -4      740
##  4  2013     1     1      555            600        -5      913
##  5  2013     1     1      557            600        -3      838
##  6  2013     1     1      558            600        -2      753
##  7  2013     1     1      558            600        -2      924
##  8  2013     1     1      558            600        -2      923
##  9  2013     1     1      559            559         0      702
## 10  2013     1     1      600            600         0      851
## # ... with 141,135 more rows
```

anti_join()

Anti-joins are useful for figuring out mismatches. So say you want to know the **flights** that do not have a match in **planes**.

```
mismatch <- flights %>%
          anti_join(planes, by = "tailnum")
mismatch[,1:7]
```

```
## # A tibble: 52,606 x 7
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      558            600        -2      753
##  2  2013     1     1      559            600        -1      941
##  3  2013     1     1      600            600         0      837
##  4  2013     1     1      602            605        -3      821
##  5  2013     1     1      608            600         8      807
##  6  2013     1     1      611            600        11      945
##  7  2013     1     1      623            610        13      920
##  8  2013     1     1      624            630        -6      840
##  9  2013     1     1      628            630        -2     1137
## 10  2013     1     1      629            630        -1      824
## # ... with 52,596 more rows
```

Set Operations

Set operations work with complete rows, and compare the values of every variable. For set operations, it is assumed that `x` and `y` have the same variables, and treat all observations as sets.

- `intersect(x, y)` returns only observations in both `x` and `y`.
- `union(x, y)` returns unique observations in `x` and `y`.
- `setdiff(x, y)` returns observations in `x`, but not `y`.

Say you have these datsets.

```
df1 <- tribble(
  ~x, ~y,
   1,  1,
   2,  1
)
df2 <- tribble(
  ~x, ~y,
   1,  1,
   1,  2
)
```

Set Operations

```
intersect(df1, df2)
```

```
## # A tibble: 1 x 2
##       x     y
##   <dbl> <dbl>
## 1     1     1
```

```
union(df1, df2)  #note we get 3 rows, not 4.
```

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

```
setdiff(df1, df2)
```

```
## # A tibble: 1 x 2
##       x     y
##   <dbl> <dbl>
## 1     2     1
```

```
setdiff(df2, df1)
```

```
## # A tibble: 1 x 2
##       x     y
##   <dbl> <dbl>
## 1     1     2
```