COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

DSFBA: Data Wrangling

Data Science for Business Analytics

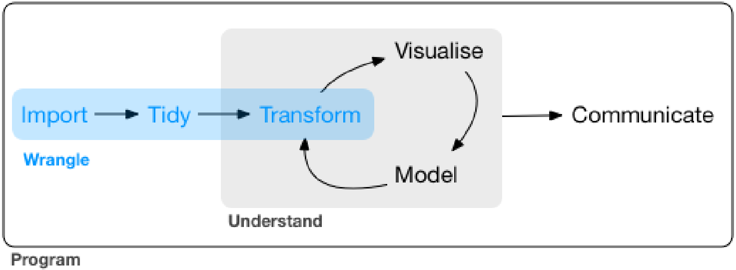
Thibault Vatter
Department of Statistics, Columbia University

10/13/2021

Outline

Today

Columbia University
in the City of New York



Most of the material (e.g., the picture above) is borrowed from

R for data science

A grammar of data manipulation

- When working with data you must:
  - Figure out what you want to do.
  - Describe those tasks as a computer program.
  - Execute the program.
- The `dplyr` package makes this fast and easy with 5 verbs!
  - `filter()`: select observations based on their values.
  - `arrange()`: reorder the observations.
  - `select()`: select variables based on their names.
  - `mutate()`: add variables as functions of existing variables.
  - `summarize()`: collapse many values down to a single summary.
- Two important features:
  - Verbs can be used with `group_by()` to operate groupwise.
  - Verbs work similarly...
    1. First argument: a data frame.
    2. Other arguments: what to do with it using variable names.
    3. Result: a new data frame.

nycflights13

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

All 336,776 flights that departed from NYC in 2013 (US BTS):

```
nycflights13::flights
#> # A tibble: 336,776 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>
#>  1  2013     1     1      517            515         2      830
#>  2  2013     1     1      533            529         4      850
#>  3  2013     1     1      542            540         2      923
#>  4  2013     1     1      544            545        -1     1004
#>  5  2013     1     1      554            600        -6      812
#>  6  2013     1     1      554            558        -4      740
#>  7  2013     1     1      555            600        -5      913
#>  8  2013     1     1      557            600        -3      709
#>  9  2013     1     1      557            600        -3      838
#> 10  2013     1     1      558            600        -2      753
#> # ... with 336,766 more rows, and 12 more variables:
#> #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#> #   time_hour <dttm>
```

What is this code doing?

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```r
a1 <- group_by(flights, year, month, day)
a2 <- select(a1, arr_delay, dep_delay)
a3 <- summarize(a2,
                arr = mean(arr_delay, na.rm = TRUE),
                dep = mean(dep_delay, na.rm = TRUE))
filter(a3, arr > 30 | dep > 30)
#> # A tibble: 49 x 5
#> # Groups:   year, month [11]
#>     year month   day   arr   dep
#>    <int> <int> <int> <dbl> <dbl>
#>  1  2013     1    16  34.2  24.6
#>  2  2013     1    31  32.6  28.7
#>  3  2013     2    11  36.3  39.1
#>  4  2013     2    27  31.3  37.8
#>  5  2013     3     8  85.9  83.5
#>  6  2013     3    18  41.3  30.1
#>  7  2013     4    10  38.4  33.0
#>  8  2013     4    12  36.0  34.8
#>  9  2013     4    18  36.0  34.9
#> 10  2013     4    19  47.9  46.1
#> # ... with 39 more rows
```

Same code (no unnecessary objects)

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
filter(summarize(select(group_by(flights, year, month, day),
           arr_delay, dep_delay),
    arr = mean(arr_delay, na.rm = TRUE),
    dep = mean(dep_delay, na.rm = TRUE)),
    arr > 30 | dep > 30)
#> # A tibble: 49 x 5
#> # Groups:   year, month [11]
#>     year month   day   arr   dep
#>    <int> <int> <int> <dbl> <dbl>
#>  1  2013     1    16  34.2  24.6
#>  2  2013     1    31  32.6  28.7
#>  3  2013     2    11  36.3  39.1
#>  4  2013     2    27  31.3  37.8
#>  5  2013     3     8  85.9  83.5
#>  6  2013     3    18  41.3  30.1
#>  7  2013     4    10  38.4  33.0
#>  8  2013     4    12  36.0  34.8
#>  9  2013     4    18  36.0  34.9
#> 10  2013     4    19  47.9  46.1
#> # ... with 39 more rows
```

. . . Or use %>%

```r
flights %>%
  group_by(year, month, day) %>%
  select(arr_delay, dep_delay) %>%
  summarize(arr = mean(arr_delay, na.rm = TRUE),
            dep = mean(dep_delay, na.rm = TRUE)) %>%
  filter(arr > 30 | dep > 30)
#> # A tibble: 49 x 5
#> # Groups:   year, month [11]
#>     year month   day   arr   dep
#>    <int> <int> <int> <dbl> <dbl>
#>  1  2013     1    16  34.2  24.6
#>  2  2013     1    31  32.6  28.7
#>  3  2013     2    11  36.3  39.1
#>  4  2013     2    27  31.3  37.8
#>  5  2013     3     8  85.9  83.5
#>  6  2013     3    18  41.3  30.1
#>  7  2013     4    10  38.4  33.0
#>  8  2013     4    12  36.0  34.8
#>  9  2013     4    18  36.0  34.9
#> 10  2013     4    19  47.9  46.1
#> # ... with 39 more rows
```

Basic piping

- `x %>% f` is equivalent to `f(x)`
- `x %>% f(y)` is equivalent to `f(x, y)`
- `x %>% f(y) %>% g(z)` is equivalent to `g(f(x, y), z)`

```
x <- 1:10
y <- x + 1
z <- y + 1
f <- function(x, y) x + y

x %>% sum
#> [1] 55
x %>% f(y)
#>  [1]  3  5  7  9 11 13 15 17 19 21
x %>% f(y) %>% f(z)
#>  [1]  6  9 12 15 18 21 24 27 30 33
```

The argument ("dot") placeholder

- `x %>% f(y, .)` is equivalent to `f(y, x)`
- `x %>% f(y, z = .)` is equivalent to `f(y, z = x)`

```
x <- 1:10
y <- 2 * x
f <- function(z, y) y / z

x %>% f(y, .)
#>  [1] 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
x %>% f(y, z = .)
#>  [1] 2 2 2 2 2 2 2 2 2 2
```

Function composition

- Each of the three options has its own strengths and weaknesses:
  - Nesting, `f(g(x))`:
    - Concise, and well suited for short sequences.
    - Longer sequences harder to read (inside out & right to left).
    - Arguments can get spread out over long distances (see Dagwood sandwich).
  - Intermediate objects, `y <- f(x); g(y)`:
    - Requires you to name intermediate objects.
    - A strength when objects are important, but a weakness when values are truly intermediate.
  - Piping, `x %>% f() %>% g()`:
    - Allows to read code in straightforward left-to-right fashion.
    - Doesn't require to name intermediate objects.
    - Only for linear sequences of transformations of a single object.
- Most code use a combination of all three styles, but...
- **Piping is more common in data analysis code!**

Outline

Tidy data

"Happy families are all alike; every unhappy family is unhappy in its own way." —- Leo Tolstoy

"Tidy datasets are all alike, but every messy dataset is messy in its own way." —- Hadley Wickham

To learn more about the underlying theory, see the Tidy Data paper.

Which representation is "best"?

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- First representation?

```
table1
#> # A tibble: 6 x 4
#>   country      year  cases population
#>   <chr>       <int>  <int>      <int>
#> 1 Afghanistan  1999    745   19987071
#> 2 Afghanistan  2000   2666   20595360
#> 3 Brazil       1999  37737  172006362
#> 4 Brazil       2000  80488  174504898
#> 5 China        1999 212258 1272915272
#> 6 China        2000 213766 1280428583
```

- Second representation?

```
table2
#> # A tibble: 12 x 4
#>    country      year type            count
#>    <chr>       <int> <chr>           <int>
#>  1 Afghanistan  1999 cases             745
#>  2 Afghanistan  1999 population   19987071
#>  3 Afghanistan  2000 cases            2666
#>  4 Afghanistan  2000 population   20595360
#>  5 Brazil       1999 cases           37737
#>  6 Brazil       1999 population  172006362
#>  7 Brazil       2000 cases           80488
#>  8 Brazil       2000 population  174504898
#>  9 China        1999 cases          212258
#> 10 China        1999 population 1272915272
#> 11 China        2000 cases          213766
#> 12 China        2000 population 1280428583
```

- Third representation?

```
table3
#> # A tibble: 6 x 3
#>   country      year rate
#> * <chr>       <int> <chr>
#> 1 Afghanistan  1999 745/19987071
#> 2 Afghanistan  2000 2666/20595360
#> 3 Brazil       1999 37737/172006362
#> 4 Brazil       2000 80488/174504898
#> 5 China        1999 212258/1272915272
#> 6 China        2000 213766/1280428583
```

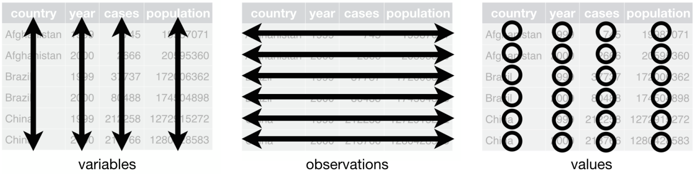
- Fourth representation?

```
table4a  # cases
#> # A tibble: 3 x 3
#>   country     `1999` `2000`
#> * <chr>        <int>  <int>
#> 1 Afghanistan    745   2666
#> 2 Brazil       37737  80488
#> 3 China       212258 213766
table4b  # population
#> # A tibble: 3 x 3
#>   country         `1999`     `2000`
#> * <chr>            <int>      <int>
#> 1 Afghanistan   19987071   20595360
#> 2 Brazil       172006362  174504898
#> 3 China       1272915272 1280428583
```

What makes a dataset tidy?

Three interrelated rules:

- Each variable must have its own column.
- Each observation must have its own row.
- Each value must have its own cell.



Because it's impossible to only satisfy two of the three:

- Put each dataset in a tibble.
- Put each variable in a column.

Why ensure that your data is tidy?

- Why?
  - With consistent data structure, it's easier to learn the tools that work with it because they have an underlying uniformity.
  - Placing variables in columns allows R's vectorized nature to shine.
- Tidy data principles seem obvious, BUT:
  - Most people aren't familiar with them.
  - Data often organized to facilitate something different than analysis.
- Hence, you'll most likely need to do some tidying.

The two steps of tidying

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- Figure out what the variables and observations are.
- Resolve one of two common problems:
  - One variable might be spread across multiple columns.
  - One observation might be scattered across multiple rows.

... To fix these problems, you'll need `pivot_longer()` and `pivot_wider()`.

Longer with `pivot_wider()`


```r
table4a
#> # A tibble: 3 x 3
#>   country     `1999` `2000`
#> * <chr>        <int>  <int>
#> 1 Afghanistan    745   2666
#> 2 Brazil       37737  80488
#> 3 China       212258 213766
```

```r
table4a %>%
  pivot_longer(c(`1999`, `2000`),
               names_to = "year",
               values_to = "cases")
#> # A tibble: 6 x 3
#>   country     year   cases
#>   <chr>       <chr>  <int>
#> 1 Afghanistan 1999     745
#> 2 Afghanistan 2000    2666
#> 3 Brazil      1999   37737
#> 4 Brazil      2000   80488
#> 5 China       1999  212258
#> 6 China       2000  213766
```

Wider with `pivot_wider()`

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```r
table2
#> # A tibble: 12 x 4
#>    country      year type            count
#>    <chr>       <int> <chr>           <int>
#>  1 Afghanistan  1999 cases             745
#>  2 Afghanistan  1999 population   19987071
#>  3 Afghanistan  2000 cases            2666
#>  4 Afghanistan  2000 population   20595360
#>  5 Brazil       1999 cases           37737
#>  6 Brazil       1999 population  172006362
#>  7 Brazil       2000 cases           80488
#>  8 Brazil       2000 population  174504898
#>  9 China        1999 cases          212258
#> 10 China        1999 population 1272915272
#> 11 China        2000 cases          213766
#> 12 China        2000 population 1280428583
```

```r
table2 %>%
    pivot_wider(names_from = type,
                values_from = count)
#> # A tibble: 6 x 4
#>   country      year  cases population
#>   <chr>       <int>  <int>      <int>
#> 1 Afghanistan  1999    745   19987071
#> 2 Afghanistan  2000   2666   20595360
#> 3 Brazil       1999  37737  172006362
#> 4 Brazil       2000  80488  174504898
#> 5 China        1999 212258 1272915272
#> 6 China        2000 213766 1280428583
```

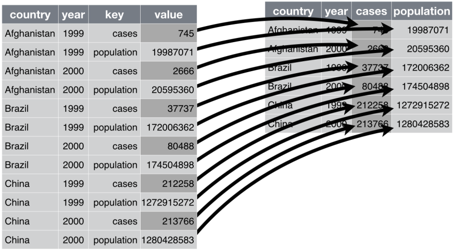


table2

Separate a column with separate()

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
table3
#> # A tibble: 6 x 3
#>   country      year rate
#> * <chr>       <int> <chr>
#> 1 Afghanistan  1999 745/19987071
#> 2 Afghanistan  2000 2666/20595360
#> 3 Brazil       1999 37737/172006362
#> 4 Brazil       2000 80488/174504898
#> 5 China        1999 212258/1272915272
#> 6 China        2000 213766/1280428583
```

```
table3 %>% separate(rate,
                    into = c("cases",
                             "population"))
#> # A tibble: 6 x 4
#>   country      year cases  population
#>   <chr>       <int> <chr>  <chr>
#> 1 Afghanistan  1999 745    19987071
#> 2 Afghanistan  2000 2666   20595360
#> 3 Brazil       1999 37737  172006362
#> 4 Brazil       2000 80488  174504898
#> 5 China        1999 212258 1272915272
#> 6 China        2000 213766 1280428583
```

country	year	rate
Afghanistan	1999	**745** / 19987071
Afghanistan	2000	**2666** / 20595360
Brazil	1999	**37737** / 172006362
Brazil	2000	**80488** / 174504898
China	1999	**212258** / 1272915272
China	2000	**213766** / 1280428583

table3

country	year	cases	population
Afghanistan	1999	**745**	19987071
Afghanistan	2000	**2666**	20595360
Brazil	1999	**37737**	172006362
Brazil	2000	**80488**	174504898
China	1999	**212258**	1272915272
China	2000	**213766**	1280428583

separate() using convert = TRUE


```r
table3 %>%
  separate(rate, into = c("cases", "population"), convert = TRUE)
#> # A tibble: 6 x 4
#>   country      year  cases population
#>   <chr>       <int>  <int>      <int>
#> 1 Afghanistan  1999    745   19987071
#> 2 Afghanistan  2000   2666   20595360
#> 3 Brazil       1999  37737  172006362
#> 4 Brazil       2000  80488  174504898
#> 5 China        1999 212258 1272915272
#> 6 China        2000 213766 1280428583
```

Unite two columns with `unite()`

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
table5
#> # A tibble: 6 x 4
#>   country     century year  rate
#> * <chr>       <chr>   <chr> <chr>
#> 1 Afghanistan 19      99    745/19987071
#> 2 Afghanistan 20      00    2666/20595360
#> 3 Brazil      19      99    37737/172006362
#> 4 Brazil      20      00    80488/174504898
#> 5 China       19      99    212258/1272915272
#> 6 China       20      00    213766/1280428583
```

```
table5 %>%
  unite(new, century, year, sep = "")
#> # A tibble: 6 x 3
#>   country     new   rate
#>   <chr>       <chr> <chr>
#> 1 Afghanistan 1999  745/19987071
#> 2 Afghanistan 2000  2666/20595360
#> 3 Brazil      1999  37737/172006362
#> 4 Brazil      2000  80488/174504898
#> 5 China       1999  212258/1272915272
#> 6 China       2000  213766/1280428583
```

country	year	rate
Afghanistan	1999	745 / 19987071
Afghanistan	2000	2666 / 20595360
Brazil	1999	37737 / 172006362
Brazil	2000	80488 / 174504898
China	1999	212258 / 1272915272
China	2000	213766 / 1280428583

country	century	year	rate
Afghanistan	19	99	745 / 19987071
Afghanistan	20	0	2666 / 20595360
Brazil	19	99	37737 / 172006362
Brazil	20	0	80488 / 174504898
China	19	99	212258 / 1272915272
China	20	0	213766 / 1280428583

table6

Missing values and tidy data

- A value can be missing in one of two possible ways:
  - **Explicitly**, i.e. flagged with `NA`.
  - **Implicitly**, i.e. simply not present in the data.

"An explicit missing value is the presence of an absence; an implicit missing value is the absence of a presence." Hadley Wickham

- Are there missing values in this dataset?

```
stocks <- tibble(
  year   = c(2015, 2015, 2015, 2015, 2016, 2016, 2016),
  qtr    = c(   1,    2,    3,    4,    2,    3,    4),
  return = c(1.88, 0.59, 0.35,   NA, 0.92, 0.17, 2.66)
)
```

Implicit to explicit and conversly

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- Implicit to explicit by pivoting:

```
stocks %>%
  pivot_wider(names_from = year,
              values_from = return)
#> # A tibble: 4 x 3
#>     qtr `2015` `2016`
#>   <dbl>  <dbl>  <dbl>
#> 1     1   1.88  NA
#> 2     2   0.59   0.92
#> 3     3   0.35   0.17
#> 4     4  NA      2.66
```

- Implicit to explicit using `complete`:

```
stocks %>% complete(year, qtr)
#> # A tibble: 8 x 3
#>    year   qtr return
#>   <dbl> <dbl>  <dbl>
#> 1  2015     1   1.88
#> 2  2015     2   0.59
#> 3  2015     3   0.35
#> 4  2015     4  NA
#> 5  2016     1  NA
#> 6  2016     2   0.92
#> 7  2016     3   0.17
#> 8  2016     4   2.66
```

- Explicit to implicit via `drop_na()`.

Fill in missing values with `fill()`

```
treatment <- tribble(
  ~ person,             ~ treatment, ~response,
  "Derrick Whitmore", 1,           7,
  NA,                 2,           10,
  NA,                 3,           9,
  "Katherine Burke",  1,           4
)
treatment %>%
  fill(person)
#> # A tibble: 4 x 3
#>   person           treatment response
#>   <chr>                <dbl>    <dbl>
#> 1 Derrick Whitmore         1        7
#> 2 Derrick Whitmore         2       10
#> 3 Derrick Whitmore         3        9
#> 4 Katherine Burke          1        4
```

Outline

Filter rows with `filter()`

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
filter(flights, month == 1, day == 1)
#> # A tibble: 842 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>
#>  1  2013     1     1      517            515         2      830
#>  2  2013     1     1      533            529         4      850
#>  3  2013     1     1      542            540         2      923
#>  4  2013     1     1      544            545        -1     1004
#>  5  2013     1     1      554            600        -6      812
#>  6  2013     1     1      554            558        -4      740
#>  7  2013     1     1      555            600        -5      913
#>  8  2013     1     1      557            600        -3      709
#>  9  2013     1     1      557            600        -3      838
#> 10  2013     1     1      558            600        -2      753
#> # ... with 832 more rows, and 12 more variables:
#> #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#> #   time_hour <dttm>
```

Comparisons

- The standard suite: >, >=, <, <=, !=, and ==.
- Most common mistake:

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

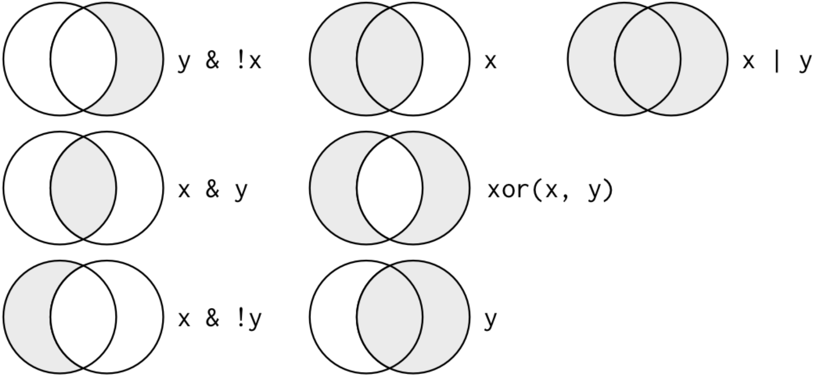
- What happens in the following?

```
sqrt(2) ^ 2 == 2
#> [1] FALSE
1/49 * 49 == 1
#> [1] FALSE
near(sqrt(2) ^ 2,  2)
#> [1] TRUE
near(1 / 49 * 49, 1)
#> [1] TRUE
```

Logical operators

Multiple arguments to `filter()` are combined with:

- & for "and"
- | for "or"
- ! for "not"



What is this code doing?

```
filter(flights, month == 11 | month == 12)
```

What is this code doing?

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
filter(flights, month == 11 | month == 12)
```

- Literally "finds all flights that departed in November or December".
- ... `filter(flights, month == 11 | 12)` ?

What is this code doing?


```
filter(flights, month == 11 | month == 12)
```

- Literally "finds all flights that departed in November or December".
- ... `filter(flights, month == 11 | 12)` ?
- No, but a solution:

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

De Morgan's law

- `!(x & y)` is the same as `!x | !y`
- `!(x | y)` is the same as `!x & !y`

```
all.equal(
  filter(flights, !(arr_delay > 120 | dep_delay > 120)),
  filter(flights, arr_delay <= 120, dep_delay <= 120)
  )
#> [1] TRUE
```

Missing values and `filter()`

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```r
df <- tibble(x = c(1, NA, 3))
filter(df, x > 1)
#> # A tibble: 1 x 1
#>       x
#>   <dbl>
#> 1     3
filter(df, is.na(x) | x > 1)
#> # A tibble: 2 x 1
#>       x
#>   <dbl>
#> 1    NA
#> 2     3
```

Outline

Arrange rows with arrange()

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
arrange(flights, year, month, day)
#> # A tibble: 336,776 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>
#>  1  2013     1     1      517            515         2      830
#>  2  2013     1     1      533            529         4      850
#>  3  2013     1     1      542            540         2      923
#>  4  2013     1     1      544            545        -1     1004
#>  5  2013     1     1      554            600        -6      812
#>  6  2013     1     1      554            558        -4      740
#>  7  2013     1     1      555            600        -5      913
#>  8  2013     1     1      557            600        -3      709
#>  9  2013     1     1      557            600        -3      838
#> 10  2013     1     1      558            600        -2      753
#> # ... with 336,766 more rows, and 12 more variables:
#> #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#> #   time_hour <dttm>
```

arrange() and desc()

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
arrange(flights, desc(arr_delay))
#> # A tibble: 336,776 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>
#>  1  2013     1     9      641            900      1301     1242
#>  2  2013     6    15     1432           1935      1137     1607
#>  3  2013     1    10     1121           1635      1126     1239
#>  4  2013     9    20     1139           1845      1014     1457
#>  5  2013     7    22      845           1600      1005     1044
#>  6  2013     4    10     1100           1900       960     1342
#>  7  2013     3    17     2321            810       911      135
#>  8  2013     7    22     2257            759       898      121
#>  9  2013    12     5      756           1700       896     1058
#> 10  2013     5     3     1133           2055       878     1250
#> # ... with 336,766 more rows, and 12 more variables:
#> #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#> #   time_hour <dttm>
```

arrange() and missing values

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
df <- tibble(x = c(5, NA, 2))
arrange(df, x)
#> # A tibble: 3 x 1
#>       x
#>   <dbl>
#> 1     2
#> 2     5
#> 3    NA
arrange(df, desc(x))
#> # A tibble: 3 x 1
#>       x
#>   <dbl>
#> 1     5
#> 2     2
#> 3    NA
```

Outline

Select columns with `select()`

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

All columns between year and day


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

All columns except from year to day

```
select(flights, -(year:day))
#> # A tibble: 336,776 x 16
#>    dep_time sched_dep_time dep_delay arr_time sched_arr_time
#>       <int>          <int>     <dbl>    <int>          <int>
#>  1      517            515         2      830            819
#>  2      533            529         4      850            830
#>  3      542            540         2      923            850
#>  4      544            545        -1     1004           1022
#>  5      554            600        -6      812            837
#>  6      554            558        -4      740            728
#>  7      555            600        -5      913            854
#>  8      557            600        -3      709            723
#>  9      557            600        -3      838            846
#> 10      558            600        -2      753            745
#> # ... with 336,766 more rows, and 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>
```

select() and everything()

```
select(flights, time_hour, air_time, everything())
#> # A tibble: 336,776 x 19
#>    time_hour           air_time  year month   day dep_time
#>    <dttm>                 <dbl> <int> <int> <int>    <int>
#>  1 2013-01-01 05:00:00      227  2013     1     1      517
#>  2 2013-01-01 05:00:00      227  2013     1     1      533
#>  3 2013-01-01 05:00:00      160  2013     1     1      542
#>  4 2013-01-01 05:00:00      183  2013     1     1      544
#>  5 2013-01-01 06:00:00      116  2013     1     1      554
#>  6 2013-01-01 05:00:00      150  2013     1     1      554
#>  7 2013-01-01 06:00:00      158  2013     1     1      555
#>  8 2013-01-01 06:00:00       53  2013     1     1      557
#>  9 2013-01-01 06:00:00      140  2013     1     1      557
#> 10 2013-01-01 06:00:00      138  2013     1     1      558
#> # ... with 336,766 more rows, and 13 more variables:
#> #   sched_dep_time <int>, 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>
```

More on select

- Helper functions you can use within `select()`:
  - `starts_with("abc")`: matches names that begin with "abc".
  - `ends_with("xyz")`: matches names that end with "xyz".
  - `contains("ijk")`: matches names that contain "ijk".
  - `matches("(.)\\1")`: selects variables that match a regular expression (this one matches any variables that contain repeated characters).
  - `num_range("x", 1:3)` matches `x1`, `x2` and `x3`.
- `select()` can be used to rename variables, but it drops all of the variables not explicitly mentioned. Instead, use `rename()`
- See `?select` for more details.

Outline

Create a narrower dataset

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
(flights_sml <- select(flights,
  ends_with("delay"),
  distance,
  air_time))
#> # A tibble: 336,776 x 4
#>    dep_delay arr_delay distance air_time
#>        <dbl>     <dbl>    <dbl>    <dbl>
#>  1         2        11     1400      227
#>  2         4        20     1416      227
#>  3         2        33     1089      160
#>  4        -1       -18     1576      183
#>  5        -6       -25      762      116
#>  6        -4        12      719      150
#>  7        -5        19     1065      158
#>  8        -3       -14      229       53
#>  9        -3        -8      944      140
#> 10        -2         8      733      138
#> # ... with 336,766 more rows
```

Add new variables with `mutate()`


```
mutate(flights_sml,
  gain = arr_delay - dep_delay,
  speed = distance / air_time * 60)
#> # A tibble: 336,776 x 6
#>    dep_delay arr_delay distance air_time  gain speed
#>        <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
#>  1         2        11     1400      227     9  370.
#>  2         4        20     1416      227    16  374.
#>  3         2        33     1089      160    31  408.
#>  4        -1       -18     1576      183   -17  517.
#>  5        -6       -25      762      116   -19  394.
#>  6        -4        12      719      150    16  288.
#>  7        -5        19     1065      158    24  404.
#>  8        -3       -14      229       53   -11  259.
#>  9        -3        -8      944      140    -5  405.
#> 10        -2         8      733      138    10  319.
#> # ... with 336,766 more rows
```

Refer to columns just created

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
mutate(flights_sml,
  gain = arr_delay - dep_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours)
#> # A tibble: 336,776 x 7
#>    dep_delay arr_delay distance air_time  gain hours gain_per_hour
#>        <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>         <dbl>
#>  1         2        11     1400      227     9 3.78           2.38
#>  2         4        20     1416      227    16 3.78           4.23
#>  3         2        33     1089      160    31 2.67          11.6
#>  4        -1       -18     1576      183   -17 3.05          -5.57
#>  5        -6       -25      762      116   -19 1.93          -9.83
#>  6        -4        12      719      150    16 2.5            6.4
#>  7        -5        19     1065      158    24 2.63           9.11
#>  8        -3       -14      229       53   -11 0.883        -12.5
#>  9        -3        -8      944      140    -5 2.33          -2.14
#> 10        -2         8      733      138    10 2.3            4.35
#> # ... with 336,766 more rows
```

transmute()


```
transmute(flights,
  gain = arr_delay - dep_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours)
#> # A tibble: 336,776 x 3
#>     gain hours gain_per_hour
#>    <dbl> <dbl>         <dbl>
#>  1     9 3.78           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 
#>  7    24 2.63           9.11
#>  8   -11 0.883        -12.5 
#>  9    -5 2.33          -2.14
#> 10    10 2.3            4.35
#> # ... with 336,766 more rows
```

Useful creation functions I

Any vectorized function would work, but frequently useful are:

- Arithmetic operators: `+`, `-`, `*`, `/`, `^`.
  - Vectorized with "recycling rules" (e.g., `air_time / 60`).
  - Useful in conjunction with aggregate functions (e.g., `x / sum(x)` or `y - mean(y)`).
- Modular arithmetic: `%/%` (integer division) and `%%` (remainder), where `x == y * (x %/% y) + (x %% y)`.
  - Allows you to break integers up into pieces (e.g., `hour = dep_time %/% 100` and `minute = dep_time %% 100`)
- Logs: `log()`, `log2()`, `log10()`.
  - Useful for data ranging across multiple orders of magnitude.
  - Convert multiplicative relationships to additive.

Useful creation functions II

- Offsets: `lead()` and `lag()`:
  - Refer to lead-/lagging values (e.g., compute running differences `x - lag(x)` or find values change `x != lag(x)`).

```
x <- 1:10
lag(x)
#>  [1] NA  1  2  3  4  5  6  7  8  9
lead(x)
#>  [1]  2  3  4  5  6  7  8  9 10 NA
```

- Cumulative aggregates: `cumsum()`, `cumprod()`, `cummin()`, `cummax()`, `cummean()`.

```
cumsum(x)
#>  [1]  1  3  6 10 15 21 28 36 45 55
cummean(x)
#>  [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5
```

Useful creation functions III

- Logical comparisons, <, <=, >, >=, !=
- Ranking functions: `min_rank()`, `row_number()`, `dense_rank()`, `percent_rank()`, `cume_dist()`, `ntile()`

```
y <- c(1, 2, 2, NA, 3, 4)
min_rank(y)
#> [1]  1  2  2 NA  4  5
min_rank(desc(y))
#> [1]  5  3  3 NA  2  1
row_number(y)
#> [1]  1  2  3 NA  4  5
dense_rank(y)
#> [1]  1  2  2 NA  3  4
percent_rank(y)
#> [1] 0.00 0.25 0.25   NA 0.75 1.00
cume_dist(y)
#> [1] 0.2 0.6 0.6  NA 0.8 1.0
```

Outline

Collapse values with `summarize()`

```r
summarize(flights, delay = mean(dep_delay, na.rm = TRUE))
#> # A tibble: 1 x 1
#>   delay
#>   <dbl>
#> 1  12.6
```

summarize() paired with group_by()

```
flights %>%
  group_by(year, month, day) %>%
  summarize(delay = mean(dep_delay, na.rm = TRUE))
#> `summarise()` has grouped output by 'year', 'month'. You can override using th
#> # A tibble: 365 x 4
#> # Groups:   year, month [12]
#>     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
#>  7  2013     1     7  5.42
#>  8  2013     1     8  2.55
#>  9  2013     1     9  2.28
#> 10  2013     1    10  2.84
#> # ... with 355 more rows
```

- To suppress the summarize info

```
options(dplyr.summarise.inform = FALSE)
```

An alternative to `na.rm`: pre-filter

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

not_cancelled %>%
  group_by(year, month, day) %>%
  summarize(mean = mean(dep_delay))
#> # A tibble: 365 x 4
#> # Groups:   year, month [12]
#>     year month   day  mean
#>    <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
#>  7  2013     1     7  5.42
#>  8  2013     1     8  2.55
#>  9  2013     1     9  2.28
#> 10  2013     1    10  2.84
#> # ... with 355 more rows
```

Useful summary functions I

- Measures of location: `mean()`, `median()`.
- Measures of spread: `sd()`, `IQR()`, `mad()`.
- Measures of rank: `min(x)`, `quantile(x, 0.25)`, `max(x)`.

```
not_cancelled %>%
  group_by(year, month, day) %>%
  summarize(first = min(dep_time), last = max(dep_time))
#> # A tibble: 365 x 5
#> # Groups:   year, month [12]
#>     year month   day first  last
#>    <int> <int> <int> <int> <int>
#>  1  2013     1     1   517  2356
#>  2  2013     1     2    42  2354
#>  3  2013     1     3    32  2349
#>  4  2013     1     4    25  2358
#>  5  2013     1     5    14  2357
#>  6  2013     1     6    16  2355
#>  7  2013     1     7    49  2359
#>  8  2013     1     8   454  2351
#>  9  2013     1     9     2  2252
#> 10  2013     1    10     3  2320
#> # ... with 355 more rows
```

Useful summary functions II

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- Measures of position: `first(x)`, `nth(x, 2)`, `last(x)`.

```
not_cancelled %>%
  group_by(year, month, day) %>%
  summarize(first_dep = first(dep_time), last_dep = last(dep_time))
#> # A tibble: 365 x 5
#> # Groups:   year, month [12]
#>     year month   day first_dep last_dep
#>    <int> <int> <int>     <int>    <int>
#>  1  2013     1     1       517     2356
#>  2  2013     1     2        42     2354
#>  3  2013     1     3        32     2349
#>  4  2013     1     4        25     2358
#>  5  2013     1     5        14     2357
#>  6  2013     1     6        16     2355
#>  7  2013     1     7        49     2359
#>  8  2013     1     8       454     2351
#>  9  2013     1     9         2     2252
#> 10  2013     1    10         3     2320
#> # ... with 355 more rows
```

Useful summary functions III

Columbia University in the City of New York

- Counts: `n(x)`, `sum(!is.na(x))`, `n_distinct(x)`.

```
not_cancelled %>%
  group_by(dest) %>%
  summarize(carriers = n_distinct(carrier)) %>%
  arrange(desc(carriers))
#> # A tibble: 104 x 2
#>    dest  carriers
#>    <chr>    <int>
#>  1 ATL          7
#>  2 BOS          7
#>  3 CLT          7
#>  4 ORD          7
#>  5 TPA          7
#>  6 AUS          6
#>  7 DCA          6
#>  8 DTW          6
#>  9 IAD          6
#> 10 MSP          6
#> # ... with 94 more rows
```

Useful summary functions IV

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- A simple helper function for counts:

```
not_cancelled %>% count(dest)
#> # A tibble: 104 x 2
#>    dest      n
#>    <chr> <int>
#>  1 ABQ     254
#>  2 ACK     265
#>  3 ALB     419
#>  4 ANC       8
#>  5 ATL   16898
#>  6 AUS    2418
#>  7 AVL     263
#>  8 BDL     412
#>  9 BGR     360
#> 10 BHM     272
#> # ... with 94 more rows
```

Useful summary functions V

- Counts with an optional weight variable:

```
not_cancelled %>% count(tailnum, wt = distance)
#> # A tibble: 4,037 x 2
#>    tailnum      n
#>    <chr>    <dbl>
#>  1 D942DN    3418
#>  2 N0EGMQ  240626
#>  3 N10156  110389
#>  4 N102UW   25722
#>  5 N103US   24619
#>  6 N104UW   25157
#>  7 N10575  141475
#>  8 N105UW   23618
#>  9 N107US   21677
#> 10 N108UW   32070
#> # ... with 4,027 more rows
```

Useful summary functions VI

- Counts of logical values: e.g., `sum(x > 10)`.

```
not_cancelled %>%
  group_by(year, month, day) %>%
  summarize(n_early = sum(dep_time < 500))
#> # A tibble: 365 x 4
#> # Groups:   year, month [12]
#>     year month   day n_early
#>    <int> <int> <int>   <int>
#>  1  2013     1     1       0
#>  2  2013     1     2       3
#>  3  2013     1     3       4
#>  4  2013     1     4       3
#>  5  2013     1     5       3
#>  6  2013     1     6       2
#>  7  2013     1     7       2
#>  8  2013     1     8       1
#>  9  2013     1     9       3
#> 10  2013     1    10       3
#> # ... with 355 more rows
```

Useful summary functions VII

- Proportions of logical values: e.g., `mean(y == 0)`.

```
not_cancelled %>%
  group_by(year, month, day) %>%
  summarize(hour_perc = mean(arr_delay > 60, na.rm = TRUE))
#> # A tibble: 365 x 4
#> # Groups:   year, month [12]
#>     year month   day hour_perc
#>    <int> <int> <int>     <dbl>
#>  1  2013     1     1    0.0722
#>  2  2013     1     2    0.0851
#>  3  2013     1     3    0.0567
#>  4  2013     1     4    0.0396
#>  5  2013     1     5    0.0349
#>  6  2013     1     6    0.0470
#>  7  2013     1     7    0.0333
#>  8  2013     1     8    0.0213
#>  9  2013     1     9    0.0202
#> 10  2013     1    10    0.0183
#> # ... with 355 more rows
```

Grouping by multiple variables I

Columbia University in the City of New York

```
daily <- group_by(flights, year, month, day)
(per_day   <- summarize(daily, flights = n()))
#> # A tibble: 365 x 4
#> # Groups:   year, month [12]
#>     year month   day flights
#>    <int> <int> <int>   <int>
#>  1  2013     1     1     842
#>  2  2013     1     2     943
#>  3  2013     1     3     914
#>  4  2013     1     4     915
#>  5  2013     1     5     720
#>  6  2013     1     6     832
#>  7  2013     1     7     933
#>  8  2013     1     8     899
#>  9  2013     1     9     902
#> 10  2013     1    10     932
#> # ... with 355 more rows
```

Grouping by multiple variables II

Columbia University in the City of New York

```
(per_month <- summarize(per_day, flights = sum(flights)))
#> # A tibble: 12 x 3
#> # Groups:   year [1]
#>     year month flights
#>    <int> <int>   <int>
#>  1  2013     1   27004
#>  2  2013     2   24951
#>  3  2013     3   28834
#>  4  2013     4   28330
#>  5  2013     5   28796
#>  6  2013     6   28243
#>  7  2013     7   29425
#>  8  2013     8   29327
#>  9  2013     9   27574
#> 10  2013    10   28889
#> 11  2013    11   27268
#> 12  2013    12   28135
(per_year  <- summarize(per_month, flights = sum(flights)))
#> # A tibble: 1 x 2
#>    year flights
#>   <int>   <int>
#> 1  2013  336776
```

Ungrouping

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
daily %>%
  ungroup() %>%                 # no longer grouped by date
  summarize(flights = n())  # all flights
#> # A tibble: 1 x 1
#>   flights
#>     <int>
#> 1  336776
```

Grouped filters

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
(popular_dests <- flights %>%
    group_by(dest) %>%
    filter(n() > 365))
#> # A tibble: 332,577 x 19
#> # Groups:   dest [77]
#>     year month   day dep_time sched_dep_time dep_delay arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>
#>  1  2013     1     1      517            515         2      830
#>  2  2013     1     1      533            529         4      850
#>  3  2013     1     1      542            540         2      923
#>  4  2013     1     1      544            545        -1     1004
#>  5  2013     1     1      554            600        -6      812
#>  6  2013     1     1      554            558        -4      740
#>  7  2013     1     1      555            600        -5      913
#>  8  2013     1     1      557            600        -3      709
#>  9  2013     1     1      557            600        -3      838
#> 10  2013     1     1      558            600        -2      753
#> # ... with 332,567 more rows, and 12 more variables:
#> #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#> #   time_hour <dttm>
```

Grouped mutates


```
popular_dests %>%
  filter(arr_delay > 0) %>%
  mutate(prop_delay = arr_delay / sum(arr_delay)) %>%
  select(year:day, dest, arr_delay, prop_delay)
#> # A tibble: 131,106 x 6
#> # Groups:   dest [77]
#>     year month   day dest  arr_delay prop_delay
#>    <int> <int> <int> <chr>     <dbl>      <dbl>
#>  1  2013     1     1 IAH          11  0.000111
#>  2  2013     1     1 IAH          20  0.000201
#>  3  2013     1     1 MIA          33  0.000235
#>  4  2013     1     1 ORD          12  0.0000424
#>  5  2013     1     1 FLL          19  0.0000938
#>  6  2013     1     1 ORD           8  0.0000283
#>  7  2013     1     1 LAX           7  0.0000344
#>  8  2013     1     1 DFW          31  0.000282
#>  9  2013     1     1 ATL          12  0.0000400
#> 10  2013     1     1 DTW          16  0.000116
#> # ... with 131,096 more rows
```

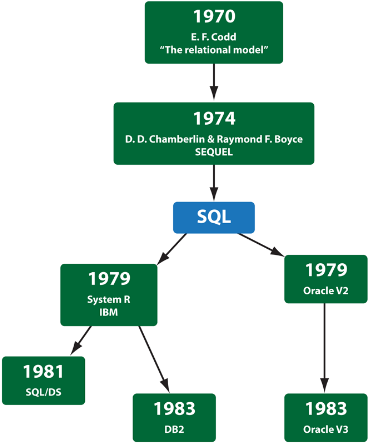
Outline

Relational data

- Until now: analysis of a single table of data.
- Typically: multiple tables of data to be combined.
  - Called **relational data**:
    - Because relations, not just the individual datasets, are important.
    - Relations are always defined for a pair of tables.
    - Relations of three or more tables are built from the relations between pairs.

RDBMS

- Common place to find relational data.
- Oracle, MySQL, Microsoft SQL Server, PostgreSQL, IBM DB2, Microsoft Access, SQLite, and others.



nycflights13::flights

- All 336,776 flights that departed from NYC in 2013 (US BTS):

```
flights
#> # A tibble: 336,776 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>
#>  1  2013     1     1      517            515         2      830
#>  2  2013     1     1      533            529         4      850
#>  3  2013     1     1      542            540         2      923
#>  4  2013     1     1      544            545        -1     1004
#>  5  2013     1     1      554            600        -6      812
#>  6  2013     1     1      554            558        -4      740
#>  7  2013     1     1      555            600        -5      913
#>  8  2013     1     1      557            600        -3      709
#>  9  2013     1     1      557            600        -3      838
#> 10  2013     1     1      558            600        -2      753
#> # ... with 336,766 more rows, and 12 more variables:
#> #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#> #   time_hour <dttm>
```

nycflights13::airlines

Columbia University in the City of New York

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

nycflights13::airports

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
airports
#> # A tibble: 1,458 x 8
#>    faa   name                 lat    lon   alt    tz dst   tzone
#>    <chr> <chr>              <dbl>  <dbl> <dbl> <dbl> <chr> <chr>
#>  1 04G   Lansdowne Airport   41.1  -80.6  1044    -5 A     America/Ne~
#>  2 06A   Moton Field Muni~   32.5  -85.7   264    -6 A     America/Ch~
#>  3 06C   Schaumburg Regio~   42.0  -88.1   801    -6 A     America/Ch~
#>  4 06N   Randall Airport     41.4  -74.4   523    -5 A     America/Ne~
#>  5 09J   Jekyll Island Ai~   31.1  -81.4    11    -5 A     America/Ne~
#>  6 0A9   Elizabethton Mun~   36.4  -82.2  1593    -5 A     America/Ne~
#>  7 0G6   Williams County ~   41.5  -84.5   730    -5 A     America/Ne~
#>  8 0G7   Finger Lakes Reg~   42.9  -76.8   492    -5 A     America/Ne~
#>  9 0P2   Shoestring Aviat~   39.8  -76.6  1000    -5 U     America/Ne~
#> 10 0S9   Jefferson County~   48.1 -123.    108    -8 A     America/Lo~
#> # ... with 1,448 more rows
```

nycflights13::planes

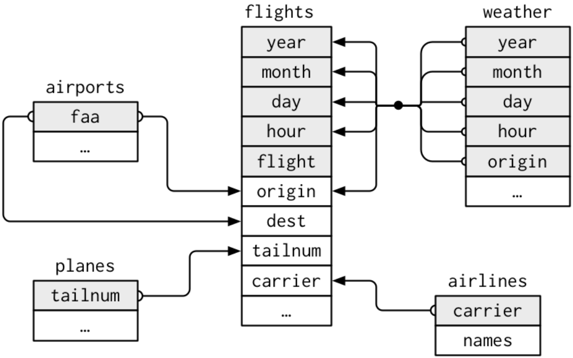
COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
planes
#> # 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 N10156   2004 Fixed~ EMBRAER      EMB-~        2    55    NA Turbo~
#>  2 N102UW   1998 Fixed~ AIRBUS INDU~ A320~        2   182    NA Turbo~
#>  3 N103US   1999 Fixed~ AIRBUS INDU~ A320~        2   182    NA Turbo~
#>  4 N104UW   1999 Fixed~ AIRBUS INDU~ A320~        2   182    NA Turbo~
#>  5 N10575   2002 Fixed~ EMBRAER      EMB-~        2    55    NA Turbo~
#>  6 N105UW   1999 Fixed~ AIRBUS INDU~ A320~        2   182    NA Turbo~
#>  7 N107US   1999 Fixed~ AIRBUS INDU~ A320~        2   182    NA Turbo~
#>  8 N108UW   1999 Fixed~ AIRBUS INDU~ A320~        2   182    NA Turbo~
#>  9 N109UW   1999 Fixed~ AIRBUS INDU~ A320~        2   182    NA Turbo~
#> 10 N110UW   1999 Fixed~ AIRBUS INDU~ A320~        2   182    NA Turbo~
#> # ... with 3,312 more rows
```

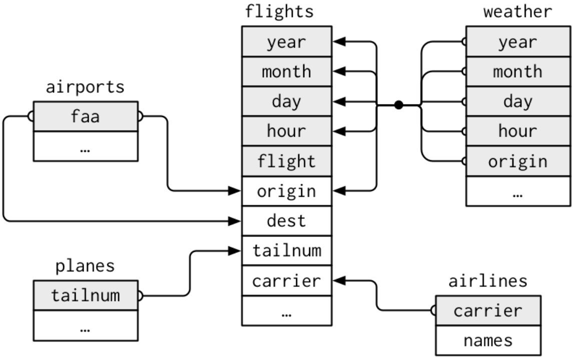
nycflights13::weather

```
weather
#> # A tibble: 26,115 x 15
#>    origin  year month   day  hour  temp  dewp humid wind_dir
#>    <chr>  <int> <int> <int> <int> <dbl> <dbl> <dbl>    <dbl>
#>  1 EWR     2013     1     1     1  39.0  26.1  59.4      270
#>  2 EWR     2013     1     1     2  39.0  27.0  61.6      250
#>  3 EWR     2013     1     1     3  39.0  28.0  64.4      240
#>  4 EWR     2013     1     1     4  39.9  28.0  62.2      250
#>  5 EWR     2013     1     1     5  39.0  28.0  64.4      260
#>  6 EWR     2013     1     1     6  37.9  28.0  67.2      240
#>  7 EWR     2013     1     1     7  39.0  28.0  64.4      240
#>  8 EWR     2013     1     1     8  39.9  28.0  62.2      250
#>  9 EWR     2013     1     1     9  39.9  28.0  62.2      260
#> 10 EWR     2013     1     1    10  41    28.0  59.6      260
#> # ... with 26,105 more rows, and 6 more variables: wind_speed <dbl>,
#> #   wind_gust <dbl>, precip <dbl>, pressure <dbl>, visib <dbl>,
#> #   time_hour <dttm>
```

nycflights13

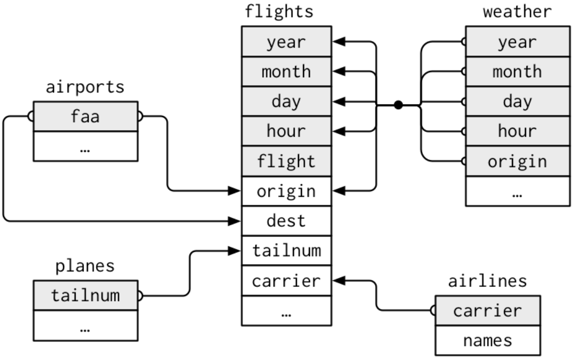


Exercise 1



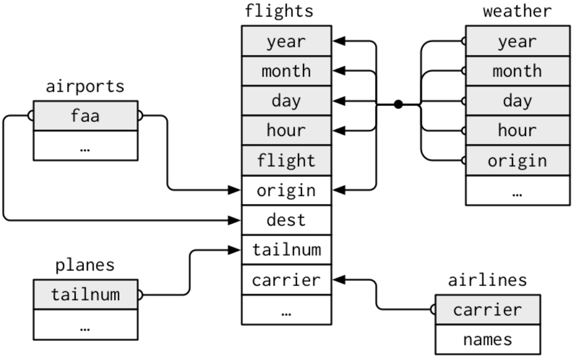
- Imagine you wanted to draw (approximately) the route each plane flies from its origin to its destination.
  - What variables would you need?
  - What tables would you need to combine?

Exercise 2



- I forgot to draw the relationship between `weather` and `airports`.
  - What is the relationship and how should it appear in the diagram?

Exercise 3



- `weather` only contains information for the origin (NYC) airports.
  - If it contained weather records for all airports in the USA, what additional relation would it define with `flights`?

Keys

- Keys:
  - Variables used to connect pair of tables.
  - Uniquely identifies an observation.
  - Can be:
    - A single variable (e.g., `tailnum` for `planes`).
    - Multiple variables (e.g., `year`, `month`, `day`, `hour`, and `origin` for `weather`).
- Two types of **keys**:
  - **Primary:** uniquely identifies an observation **in its own table.**
    - E.g., `planes$tailnum`.
  - **Foreign:** uniquely identifies an observation **in another table.**
    - E.g., `flights$tailnum`.
- Note that:
  - A variable can be both a primary key *and* a foreign key.
  - A primary key and the corresponding foreign key in another table form a **relation**.
  - Relations are typically one-to-many (e.g., flights and planes).

Is a given key primary?

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
planes %>%
  count(tailnum) %>%
  filter(n > 1)
#> # A tibble: 0 x 2
#> # ... with 2 variables: tailnum <chr>, n <int>

weather %>%
  count(year, month, day, hour, origin) %>%
  filter(n > 1)
#> # A tibble: 3 x 6
#>    year month   day  hour origin     n
#>   <int> <int> <int> <int> <chr>  <int>
#> 1  2013    11     3     1 EWR        2
#> 2  2013    11     3     1 JFK        2
#> 3  2013    11     3     1 LGA        2
```

No explicit primary key?

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

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

- Solution: add one with `mutate()` and `row_number()`.
- This is called a **surrogate key**.

Combining tables

- Two families of verbs to work with relational data:
  - **Mutating joins**
    - Add new variables to one data frame from matching observations in another.
  - **Filtering joins**
    - Filter observations from one data frame based on whether or not they match an observation in the other table.

Create a narrower dataset

```r
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
```

A simple example

Columbia University in the City of New York

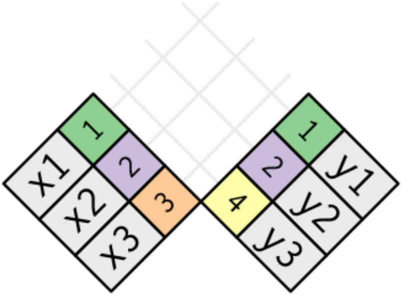
```
flights2 %>%
  select(-origin, -dest) %>%
  left_join(airlines, by = "carrier")
#> # A tibble: 336,776 x 7
#>     year month   day  hour tailnum carrier name
#>    <int> <int> <int> <dbl> <chr>   <chr>   <chr>
#>  1  2013     1     1     5 N14228  UA      United Air Lines Inc.
#>  2  2013     1     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
```

Why mutating join?

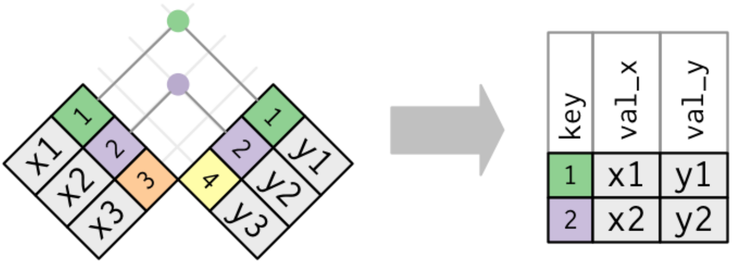

```
flights2 %>%
  select(-origin, -dest) %>%
  mutate(name = airlines$name[match(carrier, airlines$carrier)])
#> # A tibble: 336,776 x 7
#>     year month   day  hour tailnum carrier name
#>    <int> <int> <int> <dbl> <chr>   <chr>   <chr>
#>  1  2013     1     1     5 N14228  UA      United Air Lines Inc.
#>  2  2013     1     1     5 N24211  UA      United Air Lines Inc.
#>  3  2013     1     1     5 N619AA  AA      American Airlines Inc.
#>  4  2013     1     1     5 N804JB  B6      JetBlue Airways
#>  5  2013     1     1     6 N668DN  DL      Delta Air Lines Inc.
#>  6  2013     1     1     5 N39463  UA      United Air Lines Inc.
#>  7  2013     1     1     6 N516JB  B6      JetBlue Airways
#>  8  2013     1     1     6 N829AS  EV      ExpressJet Airlines Inc.
#>  9  2013     1     1     6 N593JB  B6      JetBlue Airways
#> 10  2013     1     1     6 N3ALAA  AA      American Airlines Inc.
#> # ... with 336,766 more rows
```

Understanding mutating joins

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



Inner join



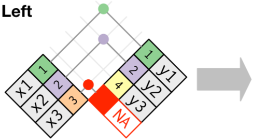

```
x %>%
  inner_join(y, by = "key")
#> # A tibble: 2 x 3
#>     key val_x val_y
#>   <dbl> <chr> <chr>
#> 1     1 x1    y1
#> 2     2 x2    y2
```

Outer joins

- **Outer joins** keep observations that appear in at least one of the tables:
  - **Left join:** keeps all observations in `x`.
  - **Right join:** keeps all observations in `y`.
  - **Full join:** keeps all observations in `x` and `y`
- They work by adding to each table an additional "virtual" observation which
  - has a key that always matches (if no other key matches),
  - and a value filled with `NA`.

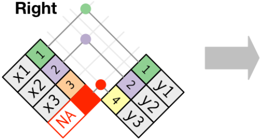
Outer joins II

Left



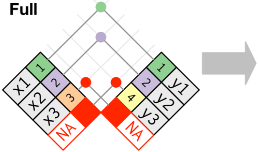
key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

Right



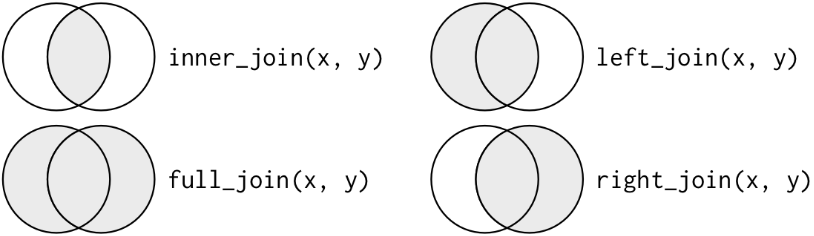
key	val_x	val_y
1	x1	y1
2	x2	y2
4	NA	y3

Full



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

A Venn diagram for joins



Duplicate keys

- Two possibilities:
  - One table has duplicate keys.
    - Useful to add in additional information as there is typically a one-to-many relationship.
  - Both tables have duplicate keys.
    - Usually an error because in neither table do the keys uniquely identify an observation.
    - When you join duplicated keys, you get all possible combinations (i.e., the Cartesian product).

One table has duplicate keys

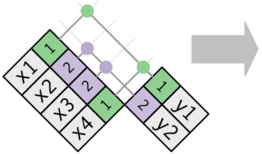
- Only `x` has duplicated keys:

```
x <- tribble(~key, ~val_x,
             1, "x1",
             2, "x2",
             2, "x3",
             1, "x4")
```

```
y <- tribble(~key, ~val_y,
             1, "y1",
             2, "y2")
```

- The join adds `val_y` to the matching rows:

```
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
```



val_x	key	val_y
x1	1	y1
x2	2	y2
x3	2	y2
x4	1	y1

Both tables have duplicate keys

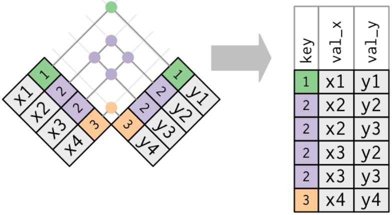
- Both `x` and `y` have duplicated keys:

```
x <- tribble(~key, ~val_x,
             1, "x1",
             2, "x2",
             2, "x3",
             3, "x4")
```

```
y <- tribble(~key, ~val_y,
             1, "y1",
             2, "y2",
             2, "y3",
             3, "y4")
```

- The joint creates all combinations:

```
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

- Default uses all variables that appear in both tables.
- Called a **natural join.**

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

Using a character vector

- Like a natural join, but uses only some of the common variables:

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

Using a named character vector

- With `by = c("a" = "b")`, `left_join` matches variable a in table `x` to variable `b` in table `y`:

```
flights2 %>%
  left_join(airports, c("dest" = "faa"))
#> # A tibble: 336,776 x 15
#>     year month   day  hour origin dest  tailnum carrier name      lat
#>    <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <chr>   <dbl>
#>  1  2013     1     1     5 EWR    IAH   N14228  UA      George~  30.0
#>  2  2013     1     1     5 LGA    IAH   N24211  UA      George~  30.0
#>  3  2013     1     1     5 JFK    MIA   N619AA  AA      Miami ~  25.8
#>  4  2013     1     1     5 JFK    BQN   N804JB  B6      <NA>     NA
#>  5  2013     1     1     6 LGA    ATL   N668DN  DL      Hartsf~  33.6
#>  6  2013     1     1     5 EWR    ORD   N39463  UA      Chicag~  42.0
#>  7  2013     1     1     6 EWR    FLL   N516JB  B6      Fort L~  26.1
#>  8  2013     1     1     6 LGA    IAD   N829AS  EV      Washin~  38.9
#>  9  2013     1     1     6 JFK    MCO   N593JB  B6      Orland~  28.4
#> 10  2013     1     1     6 LGA    ORD   N3ALAA  AA      Chicag~  42.0
#> # ... with 336,766 more rows, and 5 more variables: lon <dbl>,
#> #   alt <dbl>, tz <dbl>, dst <chr>, tzone <chr>
```

SQL is the inspiration

dplyr	SQL
`inner_join(x, y, by = "z")`	`SELECT * FROM x INNER JOIN y USING (z)`
`left_join(x, y, by = "z")`	`SELECT * FROM x LEFT OUTER JOIN y USING (z)`
`right_join(x, y, by = "z")`	`SELECT * FROM x RIGHT OUTER JOIN y USING (z)`
`full_join(x, y, by = "z")`	`SELECT * FROM x FULL OUTER JOIN y USING (z)`

- Note that:
  - "INNER" and "OUTER" are optional, and often omitted.
  - Joining different variables between the tables uses a slightly different syntax in SQL.
    - E.g. `inner_join(x, y, by = c("a" = "b"))` vs `SELECT * FROM x INNER JOIN y ON x.a = y.b`.

Filering joins

- Similar to mutating joins, but affect the observations rather than the variables:
  - `semi_join(x, y)` **keeps** all observations in `x` that have a match in `y`.
    - Useful for matching filtered summary tables back to the original rows.
  - `anti_join(x, y)` **drops** all observations in `x` that have a match in `y`.
    - Useful for diagnosing join mismatches.

Flights that went to top destinations

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

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

flights %>%
  filter(dest %in% top_dest$dest) %>%
  print(n = 5)
#> # A tibble: 141,145 x 19
#>    year month   day dep_time sched_dep_time dep_delay arr_time
#>   <int> <int> <int>    <int>          <int>     <dbl>    <int>
#> 1  2013     1     1      542            540         2      923
#> 2  2013     1     1      554            600        -6      812
#> 3  2013     1     1      554            558        -4      740
#> 4  2013     1     1      555            600        -5      913
#> 5  2013     1     1      557            600        -3      838
#> # ... with 141,140 more rows, and 12 more variables:
#> #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#> #   time_hour <dttm>
```

- How to extend to multiple variables?

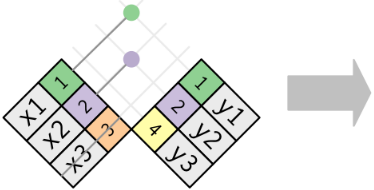
Semi-join

- Only keeps rows in `x` having a match in `y`:

```
flights %>%
  semi_join(top_dest)
#> Joining, by = "dest"
#> # A tibble: 141,145 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>
#>  1  2013     1     1      542            540         2      923
#>  2  2013     1     1      554            600        -6      812
#>  3  2013     1     1      554            558        -4      740
#>  4  2013     1     1      555            600        -5      913
#>  5  2013     1     1      557            600        -3      838
#>  6  2013     1     1      558            600        -2      753
#>  7  2013     1     1      558            600        -2      924
#>  8  2013     1     1      558            600        -2      923
#>  9  2013     1     1      559            559         0      702
#> 10  2013     1     1      600            600         0      851
#> # ... with 141,135 more rows, and 12 more variables:
#> #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#> #   time_hour <dttm>
```

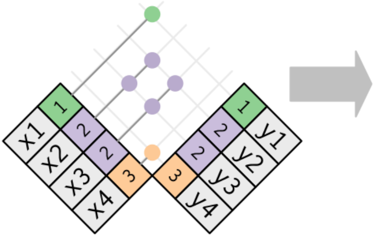
Visually understand the semi-join

- One-to-many:



key	val_x
1	x1
2	x2

- Many-to-many:



key	val_x
1	x1
2	x2
2	x3
3	x4

`flights` without a match in `planes`


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