UTSA®

ALVAREZ
College of Business
The University of Texas at San Antonio

Introduction to Programming in R

Module 5:
Data Wrangling
Part 2

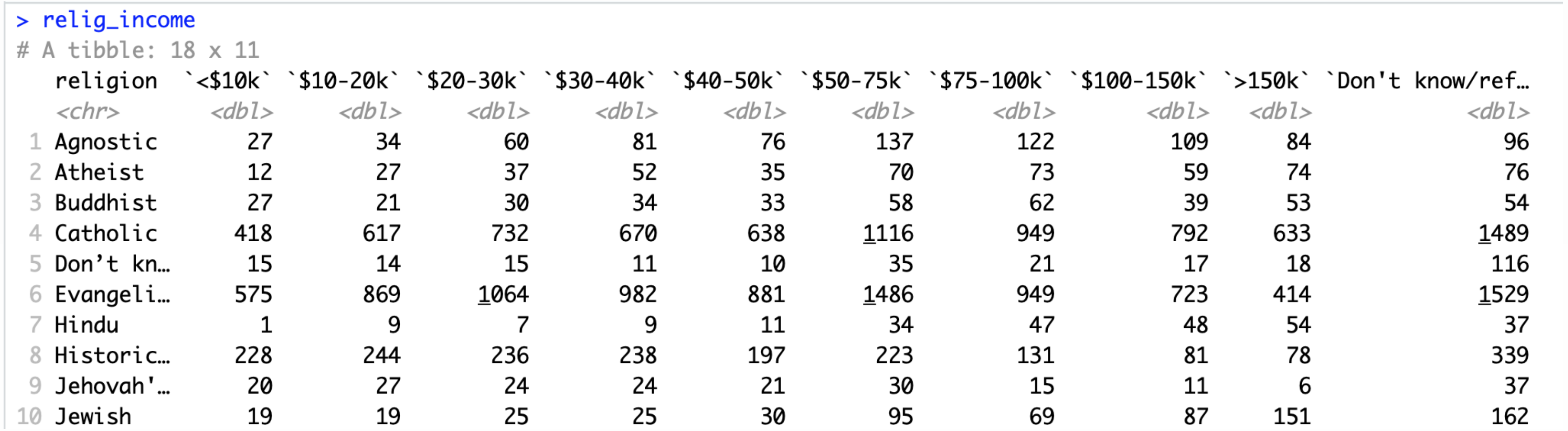
Learning Objectives

- Convert data from long to short format or vice versa.
- Separate data from one column to multiple.
- Unite multiple columns of data into one.
- Relational data, using keys to join datasets.

UTSA | ALVAREZ College of Business business.utsa.edu

Pivot: Long

- Often data may look tidy at first glance, but it is not. For e.g., some observations may be stored as variables.
- For e.g. take this built-in data on religion and income.
- Do *data("relig_income")* and *print(relig_income).*

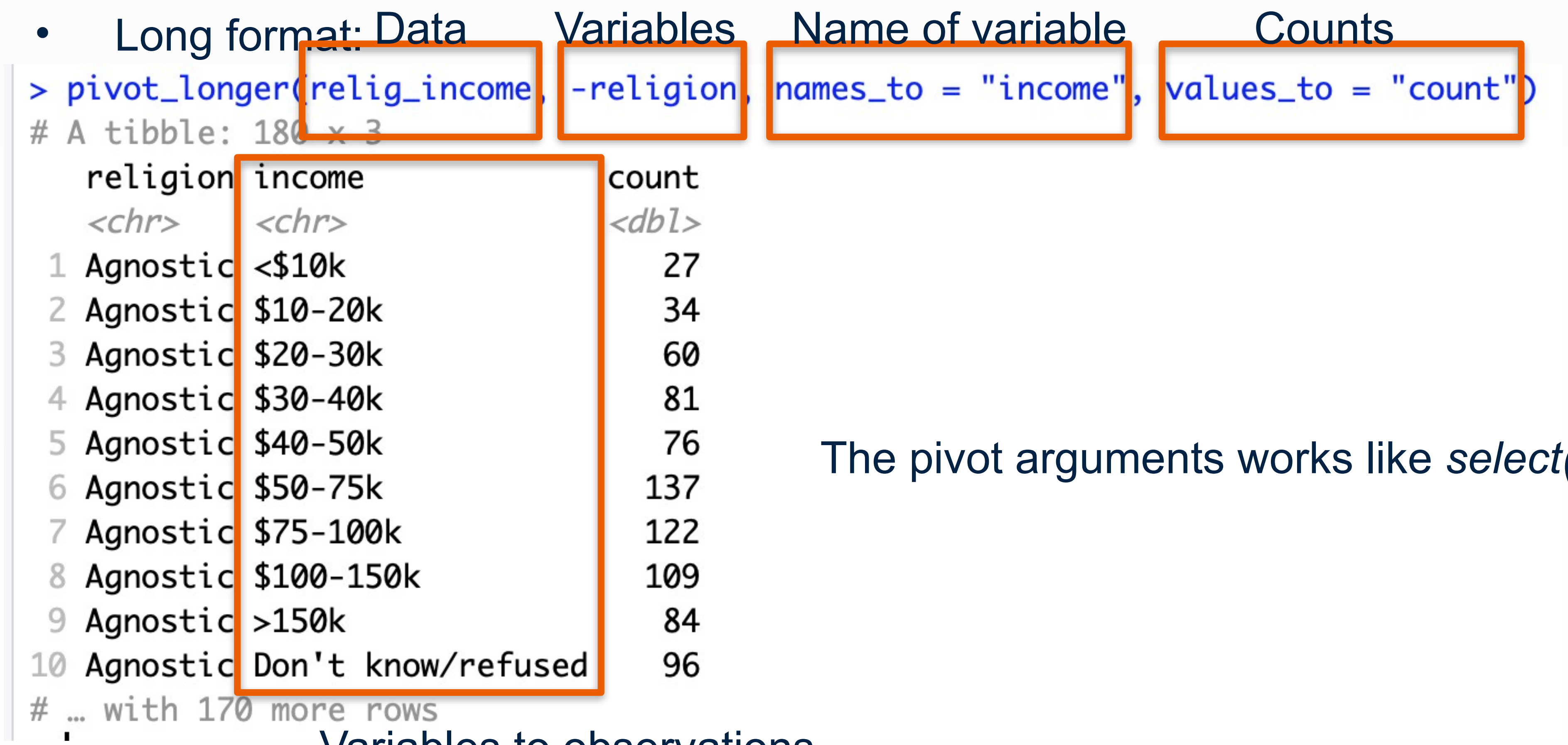


```
> relig_income
# A tibble: 18 x 11
   religion  `<$10k` `$10-20k` `$20-30k` `$30-40k` `$40-50k` `$50-75k` `$75-100k` `$100-150k` `>150k` `Don't know/ref…
   <chr>       <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>      <dbl>       <dbl>   <dbl>            <dbl>
 1 Agnostic       27        34        60        81        76       137        122         109      84               96
 2 Atheist        12        27        37        52        35        70         73          59      74               76
 3 Buddhist       27        21        30        34        33        58         62          39      53               54
 4 Catholic      418       617       732       670       638      1116        949         792     633             1489
 5 Don’t kn…      15        14        15        11        10        35         21          17      18              116
 6 Evangeli…     575       869      1064       982       881      1486        949         723     414             1529
 7 Hindu           1         9         7         9        11        34         47          48      54               37
 8 Historic…     228       244       236       238       197       223        131          81      78              339
 9 Jehovah'…      20        27        24        24        21        30         15          11       6               37
10 Jewish         19        19        25        25        30        95         69          87     151              162
```

- The income levels (variables) are observation data.
- This format of data is referred to as wider format. We can convert from wider to longer format.

Pivot: Long (cont.)

- Long format:



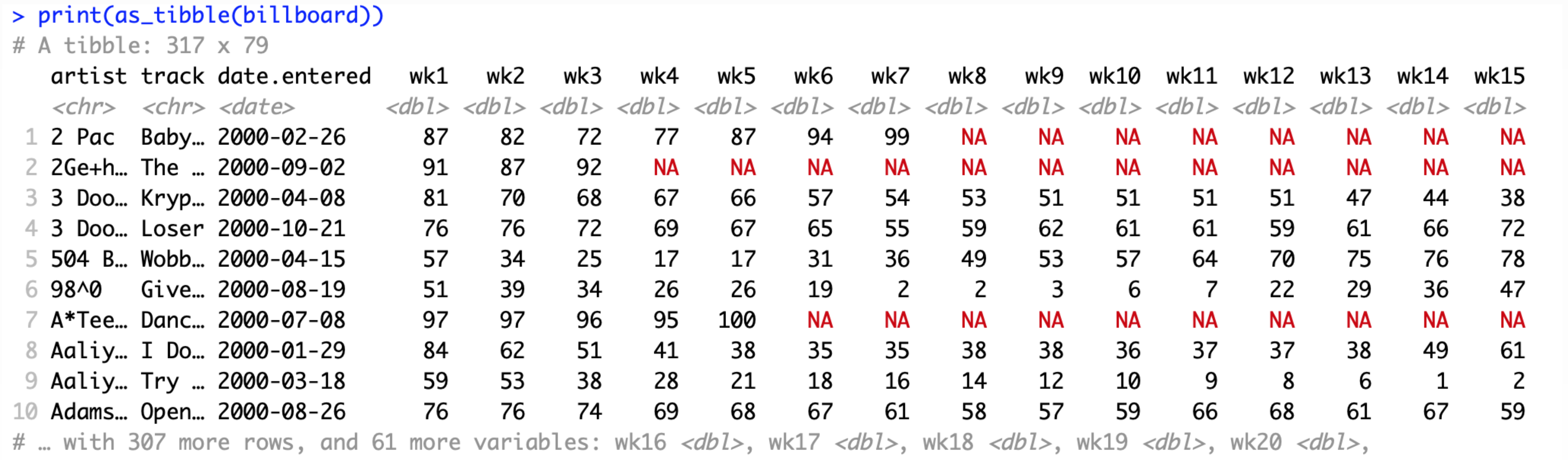
```
> pivot_longer(relig_income, -religion, names_to = "income", values_to = "count")
# A tibble: 180 x 3
   religion income             count
   <chr>    <chr>              <dbl>
 1 Agnostic <$10k                 27
 2 Agnostic $10-20k               34
 3 Agnostic $20-30k               60
 4 Agnostic $30-40k               81
 5 Agnostic $40-50k               76
 6 Agnostic $50-75k              137
 7 Agnostic $75-100k             122
 8 Agnostic $100-150k            109
 9 Agnostic >150k                 84
10 Agnostic Don't know/refused    96
# … with 170 more rows
```

The pivot arguments works like *select()*.

Variables to observations

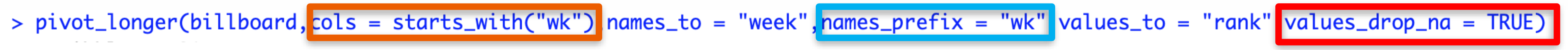
Pivot: Wide to Long

- Do *data("billboard")* and *print(billboard)*



```
> print(as_tibble(billboard))
# A tibble: 317 x 79
   artist track date.entered   wk1   wk2   wk3   wk4   wk5   wk6   wk7   wk8   wk9  wk10  wk11  wk12  wk13  wk14  wk15
   <chr>  <chr> <date>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1 2 Pac  Baby… 2000-02-26      87    82    72    77    87    94    99    NA    NA    NA    NA    NA    NA    NA    NA
 2 2Ge+h… The … 2000-09-02      91    87    92    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
 3 3 Doo… Kryp… 2000-04-08      81    70    68    67    66    57    54    53    51    51    51    51    47    44    38
 4 3 Doo… Loser 2000-10-21      76    76    72    69    67    65    55    59    62    61    61    59    61    66    72
 5 504 B… Wobb… 2000-04-15      57    34    25    17    17    31    36    49    53    57    64    70    75    76    78
 6 98^0   Give… 2000-08-19      51    39    34    26    26    19     2     2     3     6     7    22    29    36    47
 7 A*Tee… Danc… 2000-07-08      97    97    96    95   100    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
 8 Aaliy… I Do… 2000-01-29      84    62    51    41    38    35    35    38    38    36    37    37    38    49    61
 9 Aaliy… Try … 2000-03-18      59    53    38    28    21    18    16    14    12    10     9     8     6     1     2
10 Adams… Open… 2000-08-26      76    76    74    69    68    67    61    58    57    59    66    68    61    67    59
# … with 307 more rows, and 61 more variables: wk16 <dbl>, wk17 <dbl>, wk18 <dbl>, wk19 <dbl>, wk20 <dbl>,
```

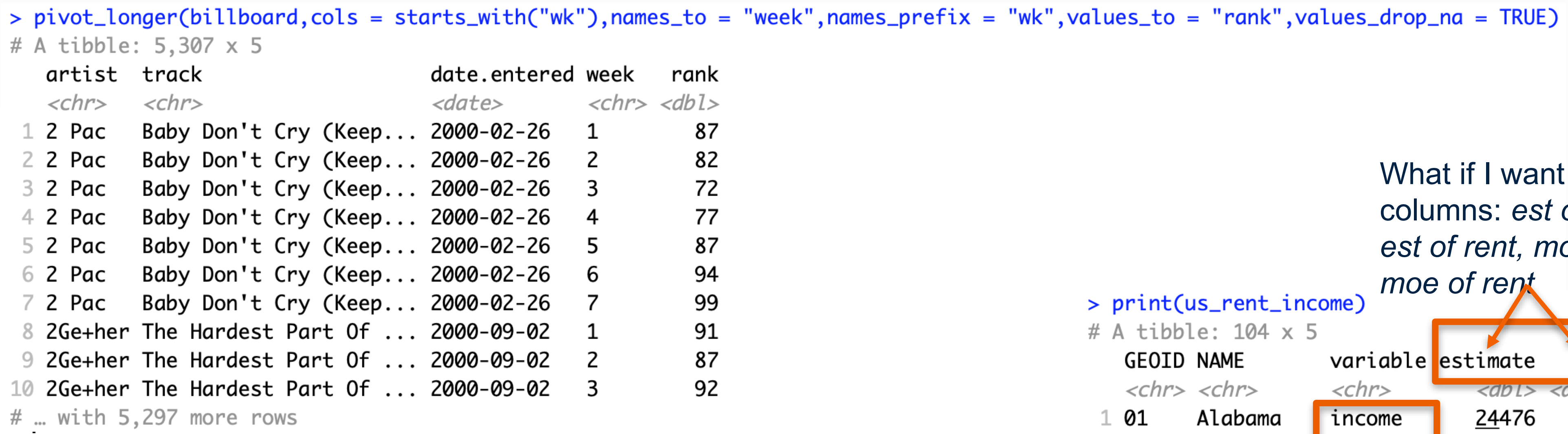
- You want to rank the songs for each week consequently, in a longer format.
- I want include all columns that start with ‘wk’, remove the prefix ‘wk’ from each variable, and ignore the NAs.



```
> pivot_longer(billboard, cols = starts_with("wk") names_to = "week", names_prefix = "wk" values_to = "rank" values_drop_na = TRUE)
```

Pivot: Long

Ignoring the NAs makes sure we stop at the week with the final ranking info.

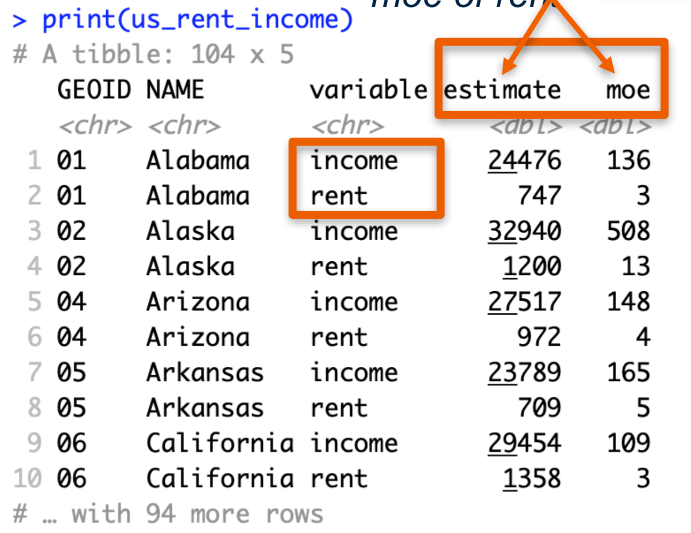


```
> pivot_longer(billboard,cols = starts_with("wk"),names_to = "week",names_prefix = "wk",values_to = "rank",values_drop_na = TRUE)
# A tibble: 5,307 x 5
   artist  track                   date.entered week   rank
   <chr>   <chr>                   <date>       <chr> <dbl>
 1 2 Pac   Baby Don't Cry (Keep... 2000-02-26   1        87
 2 2 Pac   Baby Don't Cry (Keep... 2000-02-26   2        82
 3 2 Pac   Baby Don't Cry (Keep... 2000-02-26   3        72
 4 2 Pac   Baby Don't Cry (Keep... 2000-02-26   4        77
 5 2 Pac   Baby Don't Cry (Keep... 2000-02-26   5        87
 6 2 Pac   Baby Don't Cry (Keep... 2000-02-26   6        94
 7 2 Pac   Baby Don't Cry (Keep... 2000-02-26   7        99
 8 2Ge+her The Hardest Part Of ... 2000-09-02   1        91
 9 2Ge+her The Hardest Part Of ... 2000-09-02   2        87
10 2Ge+her The Hardest Part Of ... 2000-09-02   3        92
# … with 5,297 more rows
```

- Often the data comes in a longer format and we want to convert to wider format.
- You can do this with *pivot_wider()*.
- Load the data us_rent_income for a long format example.

data("us_rent_income") and *print(us_rent_income)*

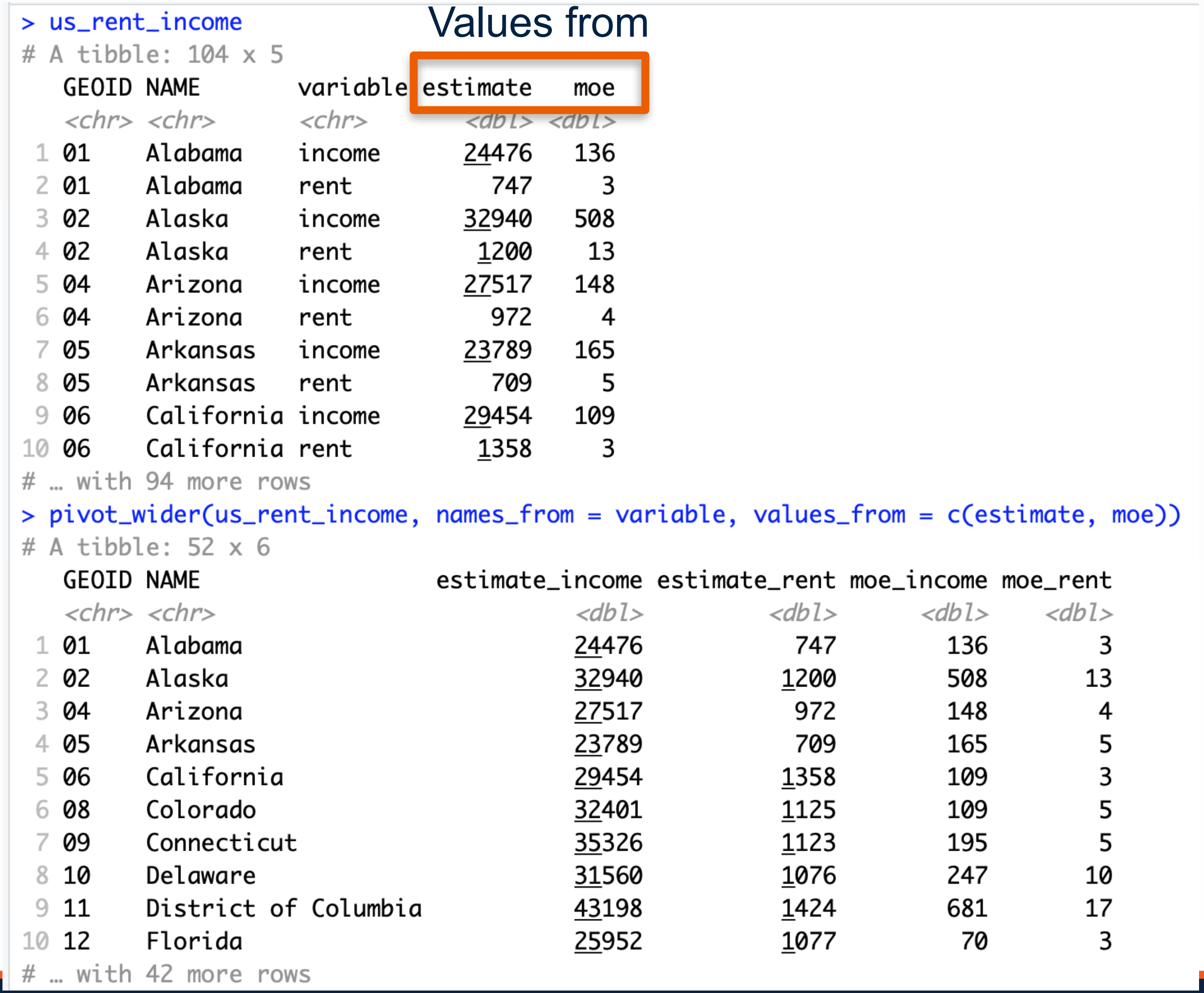
What if I want both columns: *est of income, est of rent, moe of income, moe of rent*



```
> print(us_rent_income)
# A tibble: 104 x 5
   GEOID NAME       variable estimate   moe
   <chr> <chr>      <chr>       <dbl> <dbl>
 1 01    Alabama    income      24476   136
 2 01    Alabama    rent          747     3
 3 02    Alaska     income      32940   508
 4 02    Alaska     rent         1200    13
 5 04    Arizona    income      27517   148
 6 04    Arizona    rent          972     4
 7 05    Arkansas   income      23789   165
 8 05    Arkansas   rent          709     5
 9 06    California income      29454   109
10 06    California rent         1358     3
# … with 94 more rows
```

Pivot: Long to Wide

- If there are multiple columns in the *values_from*, the column name will be appended to the front of the names_from value.
  - *estimate_income, estimate_rent, moe_income, moe_rent.*



```
> us_rent_income
# A tibble: 104 x 5
   GEOID NAME       variable estimate   moe
   <chr> <chr>      <chr>       <dbl> <dbl>
 1 01    Alabama    income      24476   136
 2 01    Alabama    rent          747     3
 3 02    Alaska     income      32940   508
 4 02    Alaska     rent         1200    13
 5 04    Arizona    income      27517   148
 6 04    Arizona    rent          972     4
 7 05    Arkansas   income      23789   165
 8 05    Arkansas   rent          709     5
 9 06    California income      29454   109
10 06    California rent         1358     3
# … with 94 more rows
> pivot_wider(us_rent_income, names_from = variable, values_from = c(estimate, moe))
# A tibble: 52 x 6
   GEOID NAME                 estimate_income estimate_rent moe_income moe_rent
   <chr> <chr>                          <dbl>         <dbl>      <dbl>    <dbl>
 1 01    Alabama                        24476           747        136        3
 2 02    Alaska                         32940          1200        508       13
 3 04    Arizona                        27517           972        148        4
 4 05    Arkansas                       23789           709        165        5
 5 06    California                     29454          1358        109        3
 6 08    Colorado                       32401          1125        109        5
 7 09    Connecticut                    35326          1123        195        5
 8 10    Delaware                       31560          1076        247       10
 9 11    District of Columbia           43198          1424        681       17
10 12    Florida                        25952          1077         70        3
# … with 42 more rows
```

Example

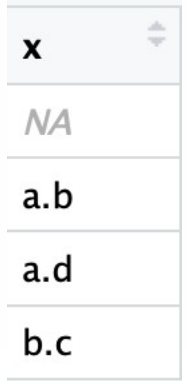
1. For *mtcars*, make the mpg value wider by expanding variables for manual and automatic transmissions, as well as, number of cylinders.
2. Reorganize the tibble with the new variables at the front.

Load the data warpbreaks – about breaks in loom (knitting).

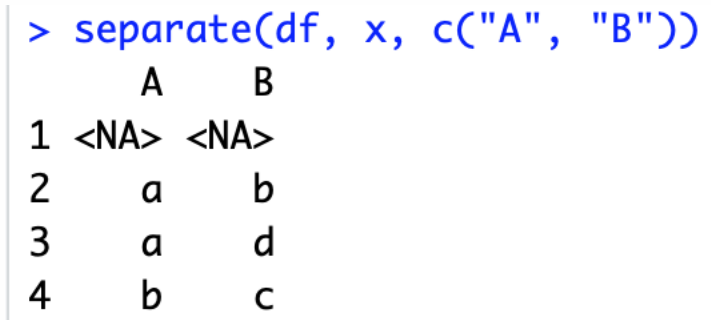
3. Make the data wider by expanding the type of wool used.
4. Are there multiple observations for each wool type and tension?
   - Used *values_fn* to impose mean on the multiple observations.

separate()

- Often, when we read in .txt files or .csv files, a single column may contain data from 2 variables, separated by a delimiter.
- The argument for *separate()* includes the name of the column to separate and the names of the new columns.
- By default, the function will separate() at non-alphanumeric character.
  - You can provide a user-specific delimiter using sep. For e.g. sep = "," will separate at a comma.
- Simple example df =



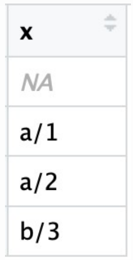
x
NA
a.b
a.d
b.c



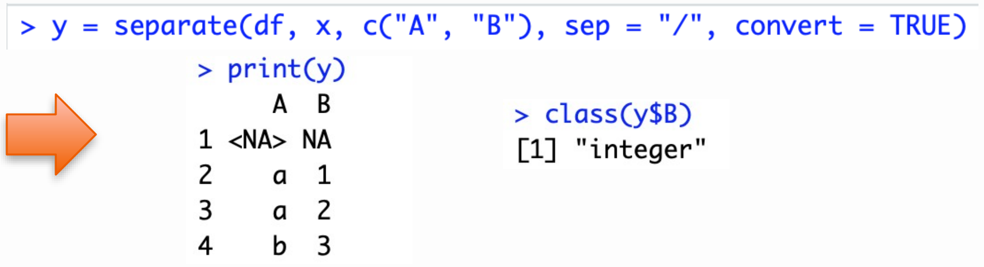
```
> separate(df, x, c("A", "B"))
     A    B
1 <NA> <NA>
2    a    b
3    a    d
4    b    c
```

separate() (cont.)

- By default, *separate()* will format the new column as a character variable.
- When we want alternatives, we can use the argument *convert*.
- Simple example: df =



x
NA
a/1
a/2
b/3

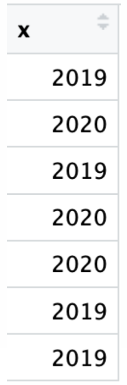


```
> y = separate(df, x, c("A", "B"), sep = "/", convert = TRUE)
> print(y)
     A  B
1 <NA> NA
2    a  1
3    a  2
4    b  3
```

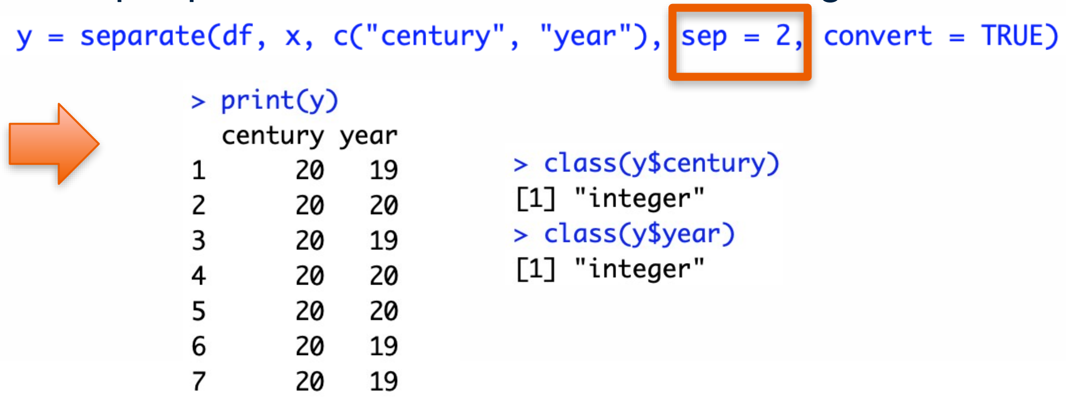
```
> class(y$B)
[1] "integer"
```

When the separated column is numeric, sep = positive value counts from left, negative from right.

- Simple example: df =



x
2019
2020
2019
2020
2020
2019
2019



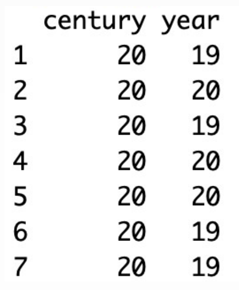
```
y = separate(df, x, c("century", "year"), sep = 2, convert = TRUE)
```

```
> print(y)
  century year
1      20   19
2      20   20
3      20   19
4      20   20
5      20   20
6      20   19
7      20   19
```

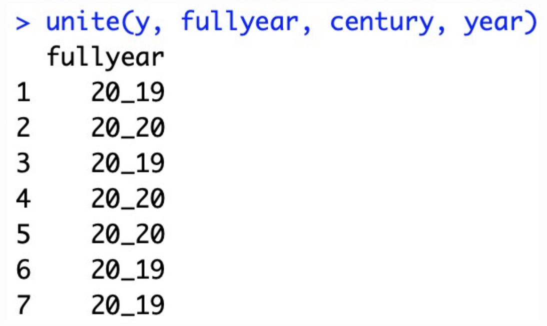
```
> class(y$century)
[1] "integer"
> class(y$year)
[1] "integer"
```

unite()

- Opposite of separate(), combines multiple columns.
- Previous simple example df =

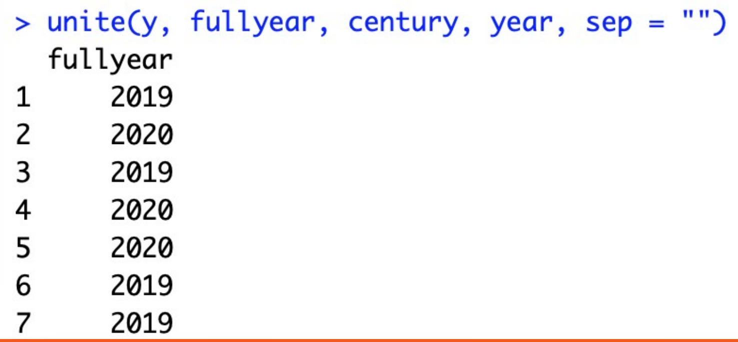


```
  century year
1      20   19
2      20   20
3      20   19
4      20   20
5      20   20
6      20   19
7      20   19
```



```
> unite(y, fullyear, century, year)
  fullyear
1    20_19
2    20_20
3    20_19
4    20_20
5    20_20
6    20_19
7    20_19
```

- The arguments of unite include the data, the new name of the combined column, followed by the columns to combine.
- By default, the separator is an underscore '_'. You can control this by *sep*.



```
> unite(y, fullyear, century, year, sep = "")
  fullyear
1     2019
2     2020
3     2019
4     2020
5     2020
6     2019
7     2019
```

Examples

Load the *esoph* data - alcohol, tobacco, and esophagus cancer data.

1. Separate the *agegp* into two variables: *MinAge* and *MaxAge*.
2. Then combine the age ranges back to *agegp* so it looks like original data.
   - How were the 75+ group handled? What can we do about the NAs?
   - You'll be happy to know that as of Dec 2019, *tidyverse* has *na.rm* option in *unite()*.
3. Make sure you get rid of the NAs in the *agegp* column.
4. Finally, make the data wider by expanding the age groups for both *ncases* and *ncontrols*.

Relational Data

- Often, multiple data frames contain related data and we need to combine and separate data from within these tables. Useful with related databases (SQL).
- Install package - *install.packages("nycflights13")* and load *library(nycflights13)*.
- Contains 5 related tables: flights, airlines, planes, weather, and airports.

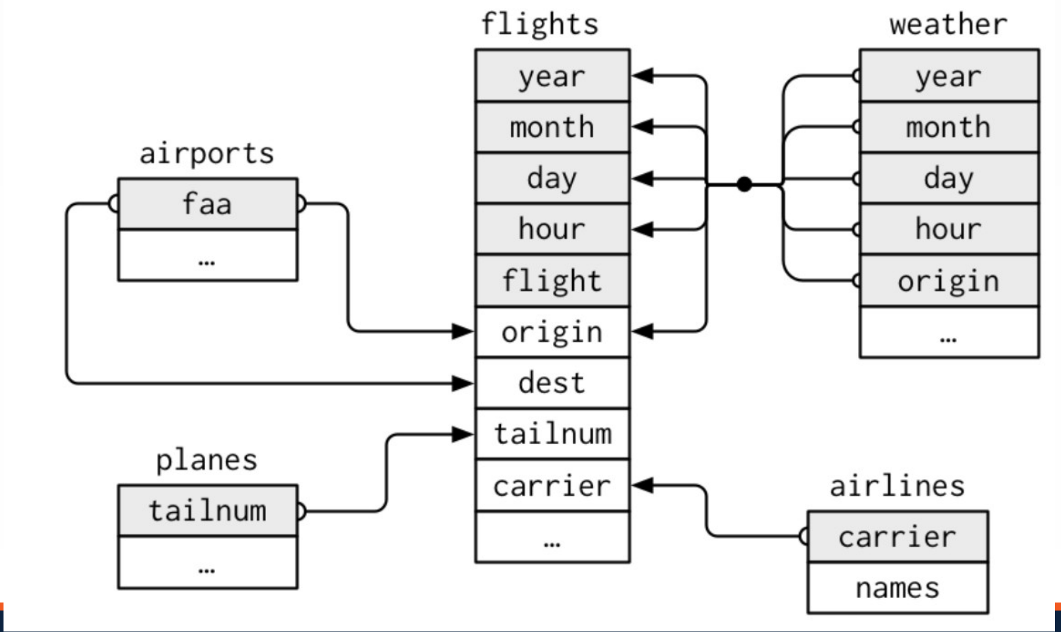
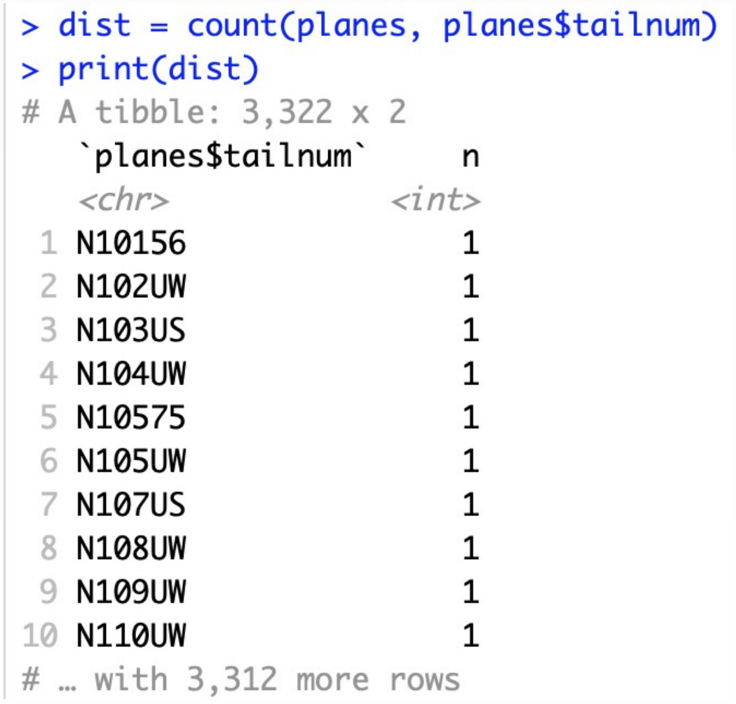


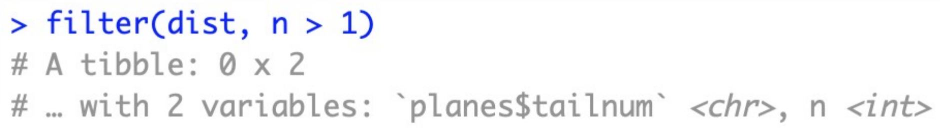
Figure from https://r4ds.had.co.nz/relational-data.html

Relational Data (cont.)

- The variable(s) that connect the tables are called keys. Unique to an observation.
- Two types of keys: primary and foreign.
    - Primary key uniquely identifies an observation in its own table. For e.g. *planes$tailnum* uniquely identifies each observation in the planes table.
    - Foreign key uniquely identifies an observation in another table. For e.g. *flights$talinum* uniquely identifies observation in planes table.
    - We can have variables that is both primary and foreign. For e.g. *weather$origin* is primary in the weather table but also uniquely identifies observations in airports table.
- Always good to double check if the primary keys are unique. Use *count()* & *filter()*.



```
> dist = count(planes, planes$tailnum)
> print(dist)
# A tibble: 3,322 x 2
   `planes$tailnum`      n
   <chr>             <int>
 1 N10156                1
 2 N102UW                1
 3 N103US                1
 4 N104UW                1
 5 N10575                1
 6 N105UW                1
 7 N107US                1
 8 N108UW                1
 9 N109UW                1
10 N110UW                1
# … with 3,312 more rows
```



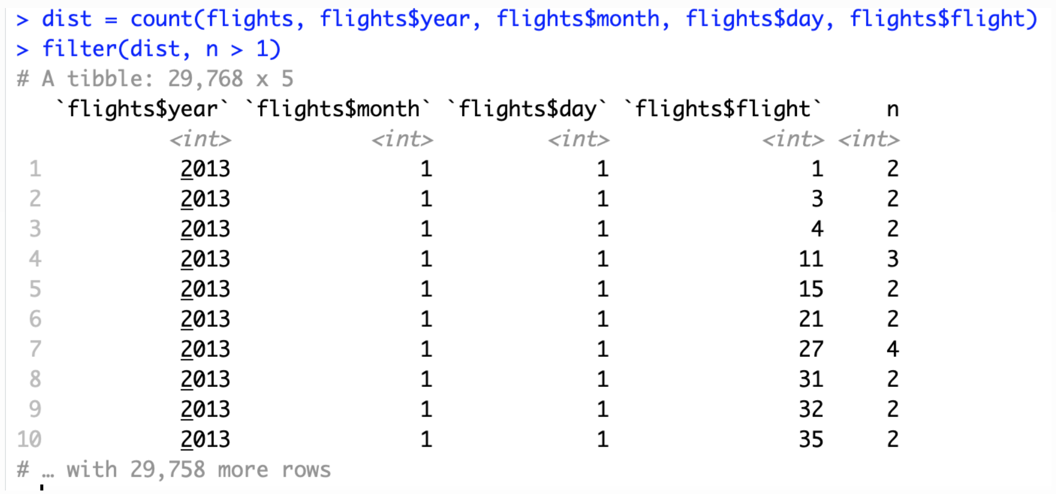
```
> filter(dist, n > 1)
# A tibble: 0 x 2
# … with 2 variables: `planes$tailnum` <chr>, n <int>
```

Relational Data (cont..)

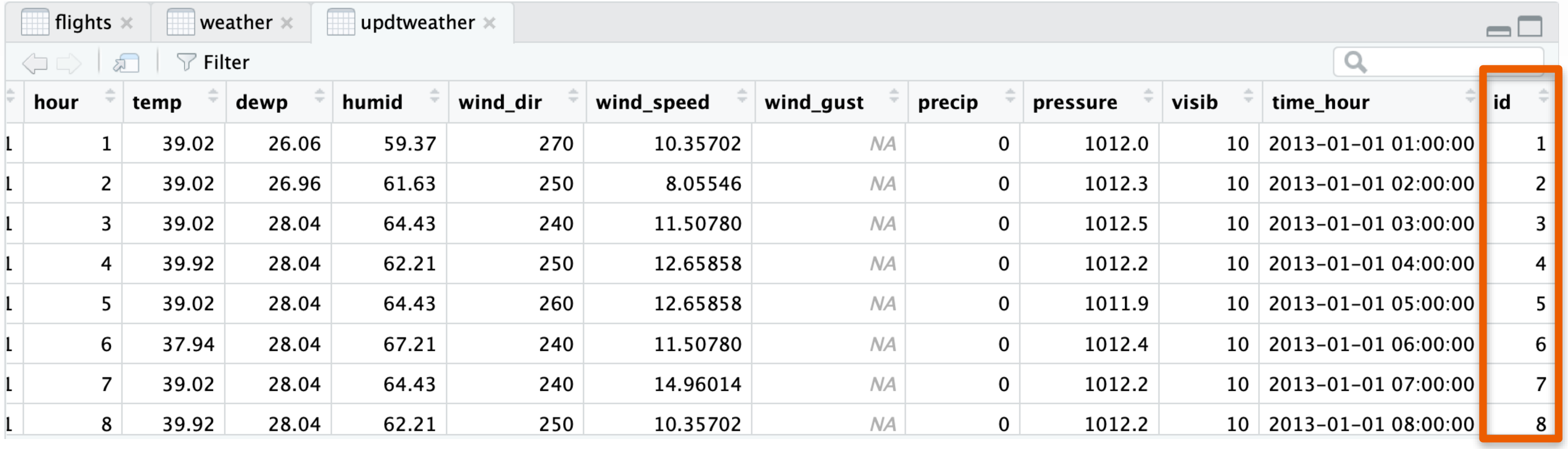
- Sometime, tables don't have primary keys. For e.g. the flights table does not have one. What about *year, month, day,* and *flight?*

- For such cases, you can use

  mutate(flights, id = row_number())



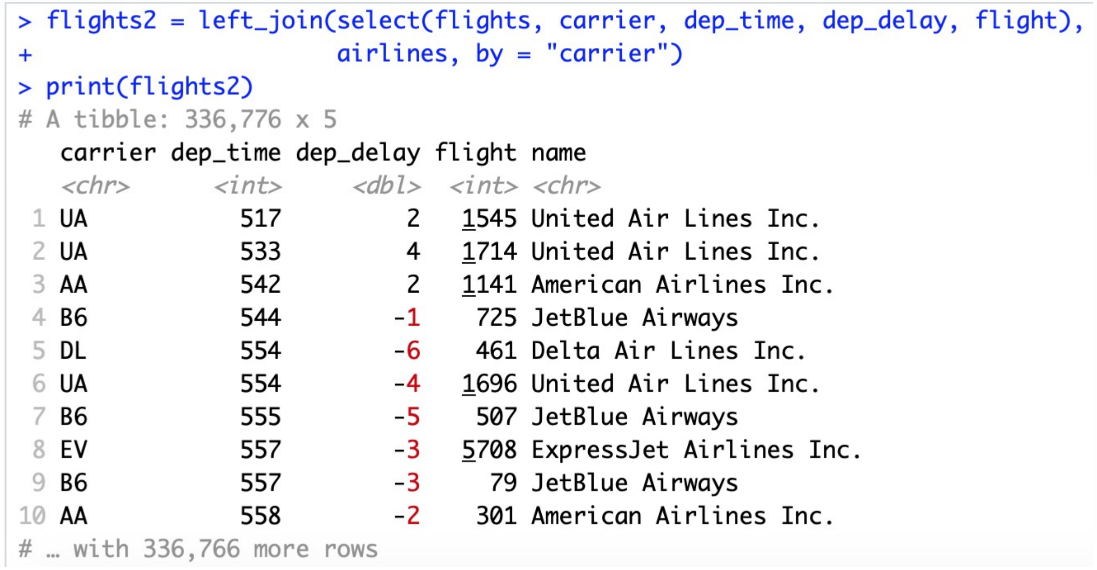
```
> dist = count(flights, flights$year, flights$month, flights$day, flights$flight)
> filter(dist, n > 1)
# A tibble: 29,768 x 5
   `flights$year` `flights$month` `flights$day` `flights$flight`     n
            <int>           <int>         <int>            <int> <int>
 1           2013               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
```



	hour	temp	dewp	humid	wind_dir	wind_speed	wind_gust	precip	pressure	visib	time_hour	id
1	1	39.02	26.06	59.37	270	10.35702	NA	0	1012.0	10	2013-01-01 01:00:00	1
1	2	39.02	26.96	61.63	250	8.05546	NA	0	1012.3	10	2013-01-01 02:00:00	2
1	3	39.02	28.04	64.43	240	11.50780	NA	0	1012.5	10	2013-01-01 03:00:00	3
1	4	39.92	28.04	62.21	250	12.65858	NA	0	1012.2	10	2013-01-01 04:00:00	4
1	5	39.02	28.04	64.43	260	12.65858	NA	0	1011.9	10	2013-01-01 05:00:00	5
1	6	37.94	28.04	67.21	240	11.50780	NA	0	1012.4	10	2013-01-01 06:00:00	6
1	7	39.02	28.04	64.43	240	14.96014	NA	0	1012.2	10	2013-01-01 07:00:00	7
1	8	39.92	28.04	62.21	250	10.35702	NA	0	1012.2	10	2013-01-01 08:00:00	8

Combining Tables: Mutating Joins

- Mutating joins allows us to combine pairs of tables. Matches observations by their keys and then copies variables across.
- Like *mutate()* – *left_join()* adds variables to the end. The argument *by* is the key.
- Suppose we wanted to put the full name of the airlines into the flights table.
- We can do this by matching by the carrier column.

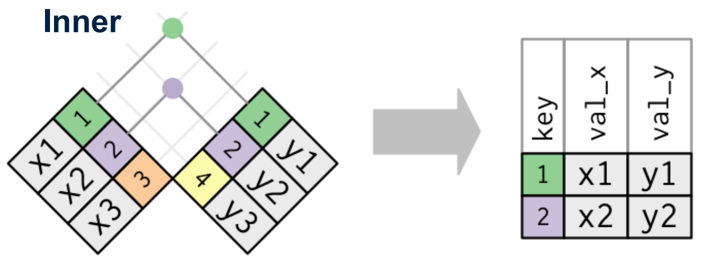


```
> flights2 = left_join(select(flights, carrier, dep_time, dep_delay, flight),
+                      airlines, by = "carrier")
> print(flights2)
# A tibble: 336,776 x 5
   carrier dep_time dep_delay flight name
   <chr>      <int>     <dbl>  <int> <chr>
 1 UA           517         2   1545 United Air Lines Inc.
 2 UA           533         4   1714 United Air Lines Inc.
 3 AA           542         2   1141 American Airlines Inc.
 4 B6           544        -1    725 JetBlue Airways
 5 DL           554        -6    461 Delta Air Lines Inc.
 6 UA           554        -4   1696 United Air Lines Inc.
 7 B6           555        -5    507 JetBlue Airways
 8 EV           557        -3   5708 ExpressJet Airlines Inc.
 9 B6           557        -3     79 JetBlue Airways
10 AA           558        -2    301 American Airlines Inc.
# … with 336,766 more rows
```

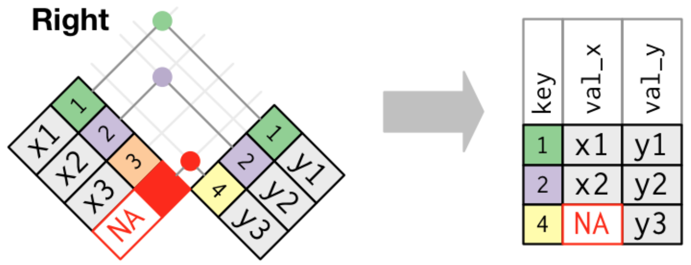
Inner & Outer Joins

- *left_join(x,y)* includes observations with keys in *x*, but not necessarily in *y*.

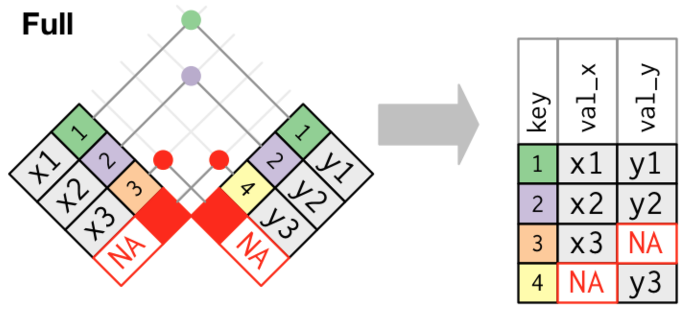
- There are alternatives:
  - *inner_join(x,y)* includes observations with keys in both *x* and *y*.
  - *right_join(x,y)* includes observations with keys in *y*, but not necessarily in *x*.
  - *full_join(x,y)* keeps all observations with keys in either in *x* or in *y*.



key	val_x	val_y
1	x1	y1
2	x2	y2



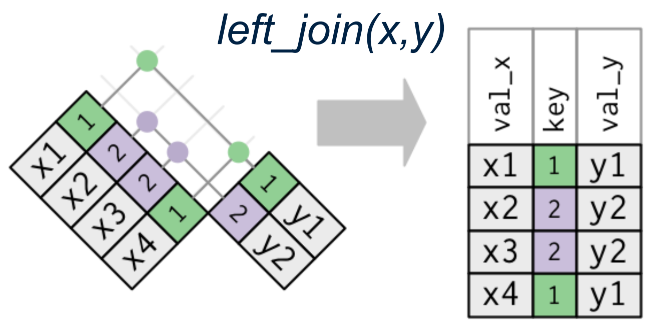
key	val_x	val_y
1	x1	y1
2	x2	y2
4	NA	y3



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

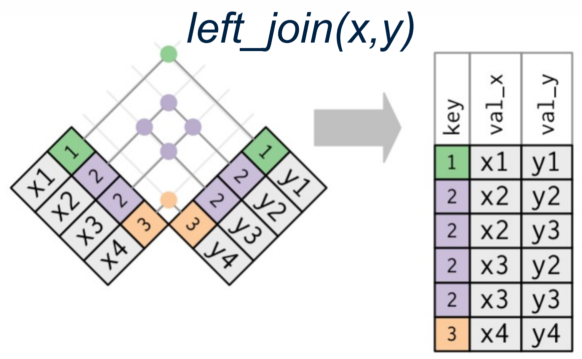
Keys

- Duplicate keys in one table – *left_join(x,y)* will keep all duplicate keys in *x*.



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

- Duplicate keys in both tables – *left_join(x,y) will create all possible combinations.*

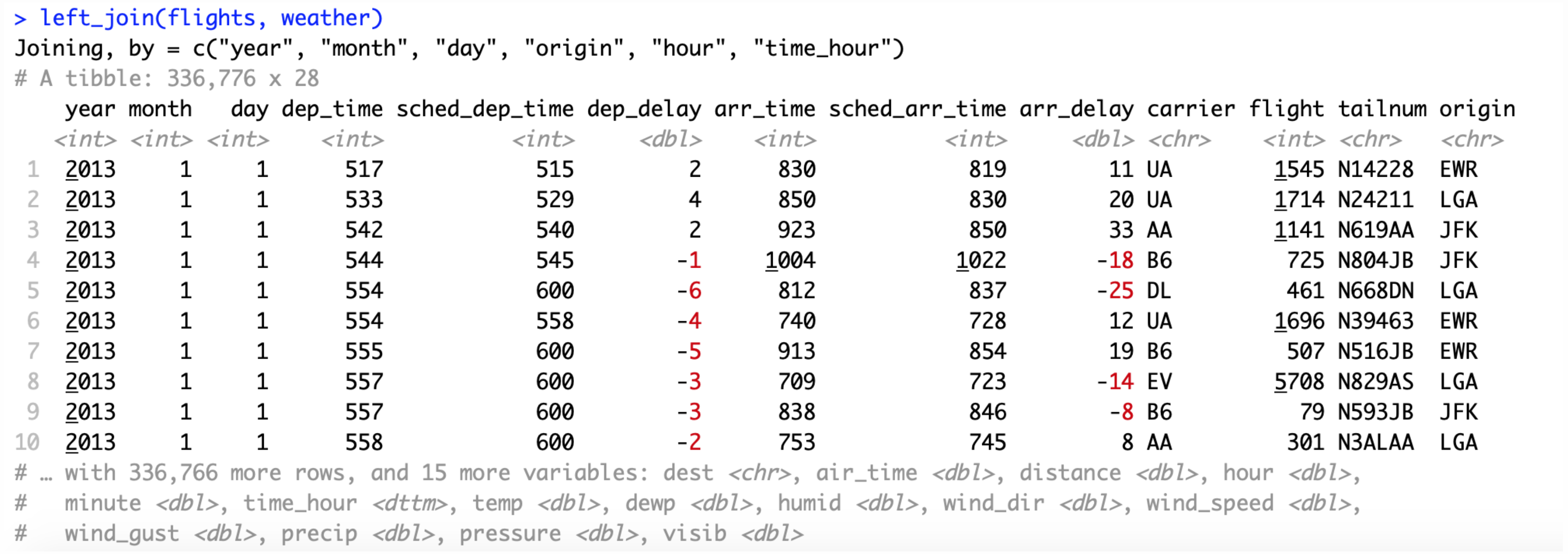


key	val_x	val_y
1	x1	y1
2	x2	y2
2	x2	y3
2	x3	y2
2	x3	y3
3	x4	y4

Figures from https://r4ds.had.co.nz/relational-data.html

Keys (cont.)

- If you ignore the keys argument the left_join() will match by common variables that exists in both tables.
  - For e.g. the flights weather tables have origin, year, month, day and hour.



```
> left_join(flights, weather)
Joining, by = c("year", "month", "day", "origin", "hour", "time_hour")
# A tibble: 336,776 x 28
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight tailnum origin
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl> <chr>    <int> <chr>   <chr> 
 1  2013     1     1      517            515         2      830            819        11 UA        1545 N14228  EWR   
 2  2013     1     1      533            529         4      850            830        20 UA        1714 N24211  LGA   
 3  2013     1     1      542            540         2      923            850        33 AA        1141 N619AA  JFK   
 4  2013     1     1      544            545        -1     1004           1022       -18 B6         725 N804JB  JFK   
 5  2013     1     1      554            600        -6      812            837       -25 DL         461 N668DN  LGA   
 6  2013     1     1      554            558        -4      740            728        12 UA        1696 N39463  EWR   
 7  2013     1     1      555            600        -5      913            854        19 B6         507 N516JB  EWR   
 8  2013     1     1      557            600        -3      709            723       -14 EV        5708 N829AS  LGA   
 9  2013     1     1      557            600        -3      838            846        -8 B6          79 N593JB  JFK   
10  2013     1     1      558            600        -2      753            745         8 AA         301 N3ALAA  LGA   
# … with 336,766 more rows, and 15 more variables: dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>, temp <dbl>, dewp <dbl>, humid <dbl>, wind_dir <dbl>, wind_speed <dbl>,
#   wind_gust <dbl>, precip <dbl>, pressure <dbl>, visib <dbl>
```

Keys (cont..)

If there are multiple keys in table *x* to join with table *y:*

- Specify using *by* = *c("a" = "b")* where column *a* in table *x* matched to column *b* in table *y.*
- For e.g. if we want to connect flights data to airport data, we need decide whether to use origin or dest to connect to faa.

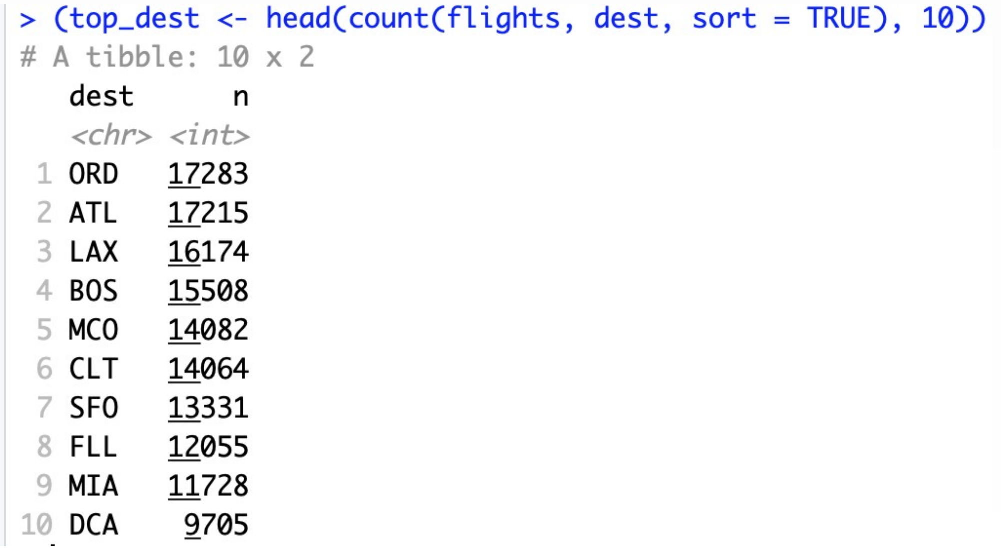
left_join(flights, airports, by = c("dest" = "faa")) will append destination airport data to the flight table.

left_join(flights, airports, by = c("origin" = "faa")) will append origin airport data to the flight table.

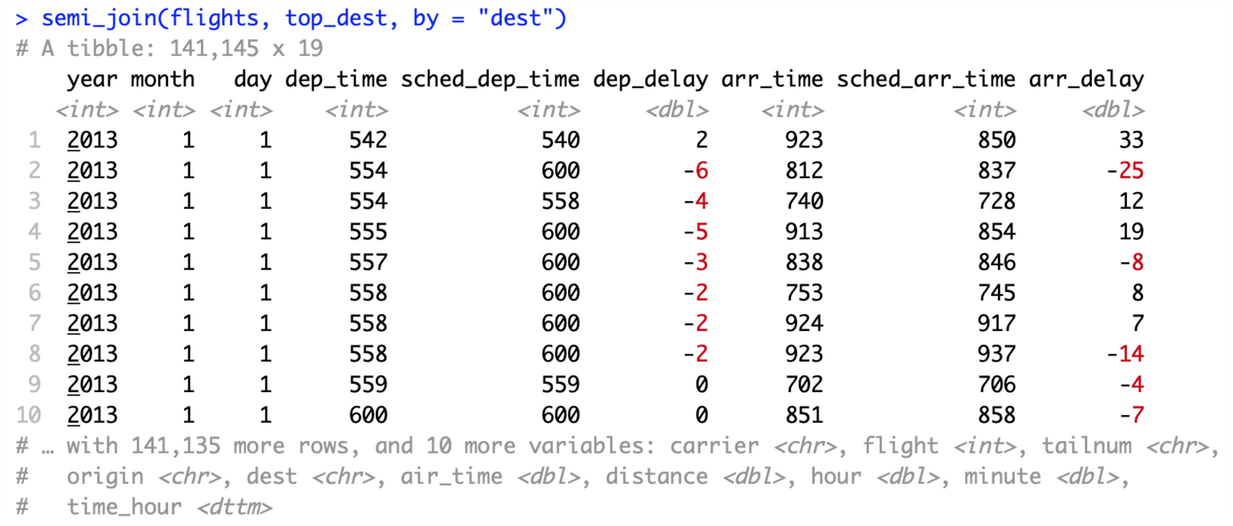
- Assign the expressions to variables and look at the last 7 columns.

Combining Tables : Filtering Joins

- Filtering joins affect the observations in the tables.
  - *semi_join(x,y)* keeps all observations in table *x* that matches *y*.
  - *anti_join(x,y)* removes all observations in table *x* that matches *y*.
- For e.g. suppose you wanted to see the data for flights flying to the top 10 destinations.



```
> (top_dest <- head(count(flights, dest, sort = TRUE), 10))
# A tibble: 10 x 2
   dest      n
   <chr> <int>
 1 ORD   17283
 2 ATL   17215
 3 LAX   16174
 4 BOS   15508
 5 MCO   14082
 6 CLT   14064
 7 SFO   13331
 8 FLL   12055
 9 MIA   11728
10 DCA    9705
```

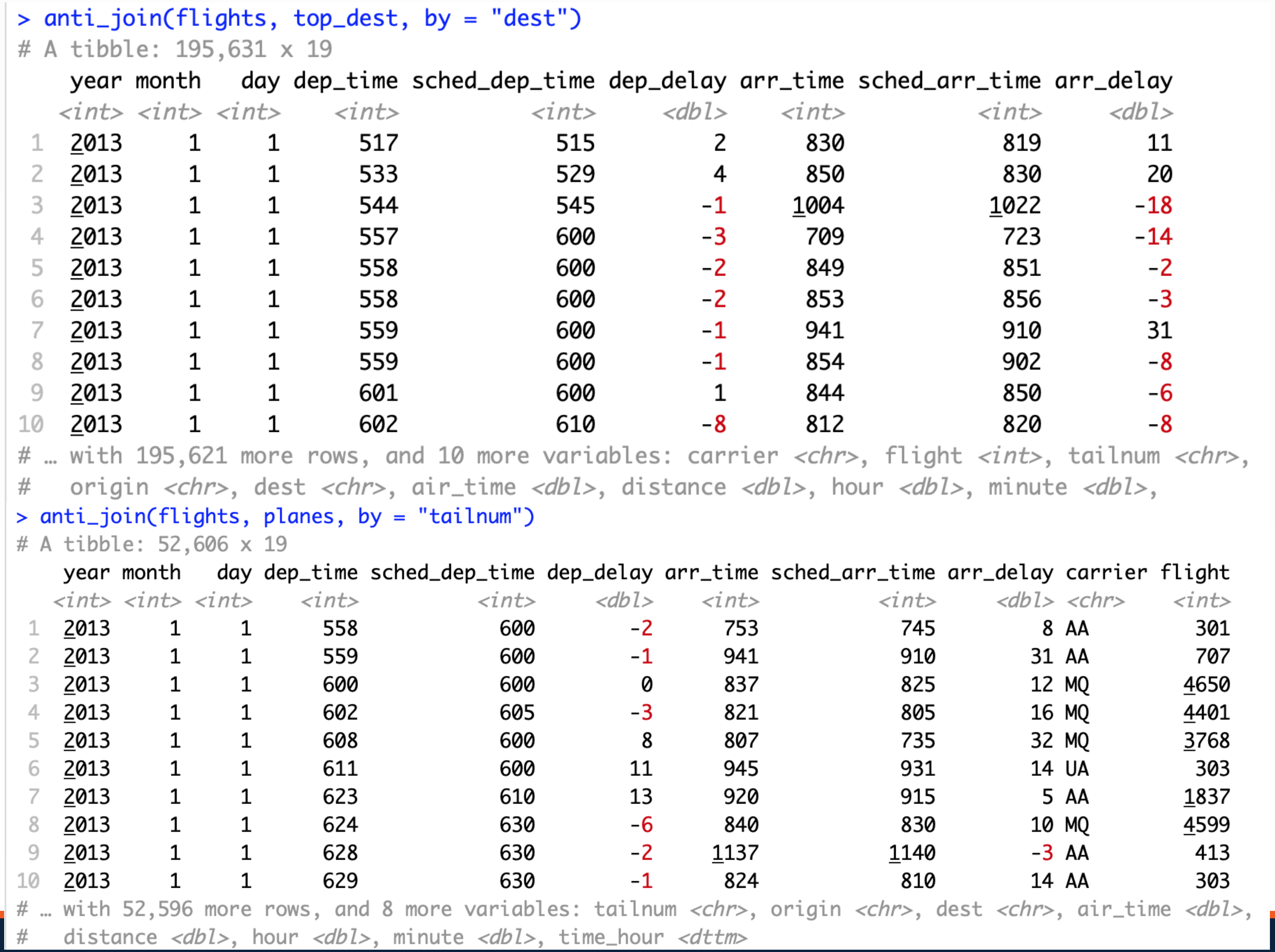


```
> semi_join(flights, top_dest, by = "dest")
# A tibble: 141,145 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
 1  2013     1     1      542            540         2      923            850        33
 2  2013     1     1      554            600        -6      812            837       -25
 3  2013     1     1      554            558        -4      740            728        12
 4  2013     1     1      555            600        -5      913            854        19
 5  2013     1     1      557            600        -3      838            846        -8
 6  2013     1     1      558            600        -2      753            745         8
 7  2013     1     1      558            600        -2      924            917         7
 8  2013     1     1      558            600        -2      923            937       -14
 9  2013     1     1      559            559         0      702            706        -4
10  2013     1     1      600            600         0      851            858        -7
# … with 141,135 more rows, and 10 more variables: carrier <chr>, flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>
```

Combining Tables : Filtering Joins (cont.)

- On the other hand, if you wanted to see the data for flights NOT flying to the top 10 destinations.

- Useful to find mismatches. For e.g. connecting flights and planes tables - find observations in flights table that don't exist in planes using "tailnum".



```
> anti_join(flights, top_dest, by = "dest")
# A tibble: 195,631 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
 1  2013     1     1      517            515         2      830            819        11
 2  2013     1     1      533            529         4      850            830        20
 3  2013     1     1      544            545        -1     1004           1022       -18
 4  2013     1     1      557            600        -3      709            723       -14
 5  2013     1     1      558            600        -2      849            851        -2
 6  2013     1     1      558            600        -2      853            856        -3
 7  2013     1     1      559            600        -1      941            910        31
 8  2013     1     1      559            600        -1      854            902        -8
 9  2013     1     1      601            600         1      844            850        -6
10  2013     1     1      602            610        -8      812            820        -8
# … with 195,621 more rows, and 10 more variables: carrier <chr>, flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
> anti_join(flights, planes, by = "tailnum")
# A tibble: 52,606 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      558            600        -2      753            745         8 AA         301
 2  2013     1     1      559            600        -1      941            910        31 AA         707
 3  2013     1     1      600            600         0      837            825        12 MQ        4650
 4  2013     1     1      602            605        -3      821            805        16 MQ        4401
 5  2013     1     1      608            600         8      807            735        32 MQ        3768
 6  2013     1     1      611            600        11      945            931        14 UA         303
 7  2013     1     1      623            610        13      920            915         5 AA        1837
 8  2013     1     1      624            630        -6      840            830        10 MQ        4599
 9  2013     1     1      628            630        -2     1137           1140        -3 AA         413
10  2013     1     1      629            630        -1      824            810        14 AA         303
# … with 52,596 more rows, and 8 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Set Operations without dplyr

- There 4 set operations in base R that could be used as well:
  - *intersect(x,y)* returns observations that are in both *x* and *y*.
  - *union(x,y)* returns unique observations in *x* and *y*.
  - setdiff(x,y) returns observations in *x* but not in *y.*
  - *merge(x,y)* same as mutate join functions –
    - *merge(x,y)* ☾ *inner_join(x,y)*
    - *merge(x,y, all.x = TRUE)* ☾ *left_join(x,y)*
    - *merge(x,y, all.y = TRUE)* ☾ *right_join(x,y)*
    - *merge(x,y, , all.x = TRUE , all.y = TRUE)* ☾ *full_join(x,y).*

Example set 1

- Load “hmda_2017_tx_all_40.csv” and “hmda_2017_tx_all_06.csv”.
- Sample 500 observations from *hmda_2017_tx_all_06.*
- Add the lien status from *hmda_2017_tx_all_40* to the sample from _06.
- Which lien status is most common?

Example set 2

Install package *Lahman* and load it. There are many data frames (df).

See *?Managers* and *?AwardsManagers*.

1. Attach the *awardID* from *AwardsManagers* to the *Managers* df.

See *?Salaries* and *?AwardsPlayers.*

2. Attach the *awardID* from *AwardsPlayers* to *Salaries* df.
3. See the top and bottom 10 salaries of players and compare awards.
4. See *?Appearances*. Are there any players who received salary in a year but never appeared for a game?
5. See *?People.* Are there any players who did not show up on salaries data, yet played games? If so, how many games? Arrange in descending order.