Tidy data, dplyr, tidyr

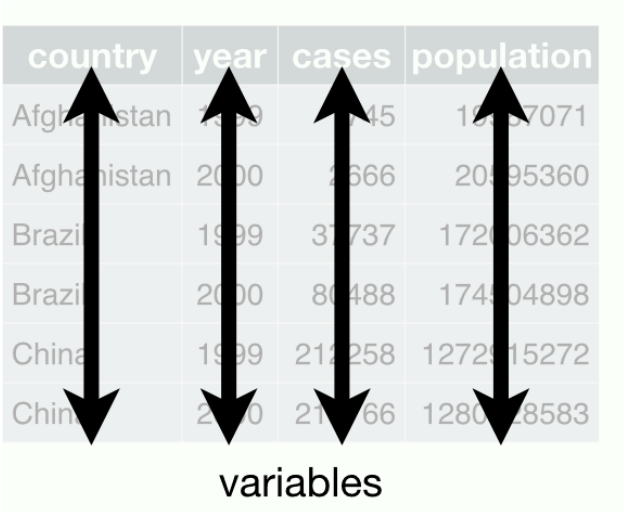
Lecture 08

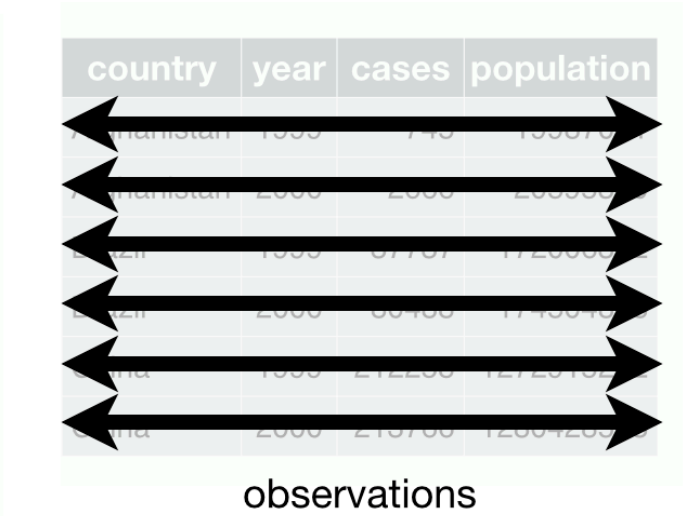
Dr. Colin Rundel

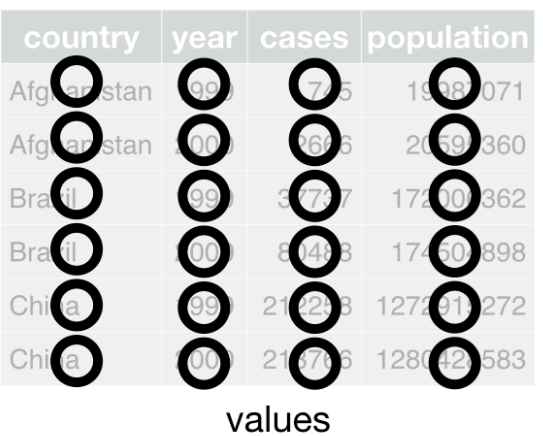
Combination of two sets of slides from previous years:

- **Tidy data & dplyr**
- **tidyr**

Tidy data


country
year
cases
population
variables


country
year
cases
population
observations


country
year
cases
population
values

Tidy vs Untidy

> Happy families are all alike; every unhappy family is unhappy in its own way
>
> — Leo Tolstoy, Anna Karenina

```
# A tibble: 317 × 7
   artist         track           date.entered   wk1   wk2   wk3   wk4
   <chr>          <chr>           <date>       <dbl> <dbl> <dbl> <dbl>
 1 2 Pac          Baby Don't Cry… 2000-02-26      87    82    72    77
 2 2Ge+her        The Hardest Pa… 2000-09-02      91    87    92    NA
 3 3 Doors Down   Kryptonite      2000-04-08      81    70    68    67
 4 3 Doors Down   Loser           2000-10-21      76    76    72    69
 5 504 Boyz       Wobble Wobble   2000-04-15      57    34    25    17
 6 98^0           Give Me Just O… 2000-08-19      51    39    34    26
 7 A*Teens        Dancing Queen   2000-07-08      97    97    96    95
 8 Aaliyah        I Don't Wanna   2000-01-29      84    62    51    41
 9 Aaliyah        Try Again       2000-03-18      59    53    38    28
10 Adams, Yolanda Open My Heart   2000-08-26      76    76    74    69
# ℹ 307 more rows
```

Is this data tidy?


TIBBLE
www.rstudio.com

Modern data frames

The tidyverse includes the tibble package that extends data frames to be a bit more modern. The core features of tibbles is to have a nicer printing method as well as being “surly” and “lazy”.

```
1 library(tibble)
```

```r
iris
```

```
    Sepal.Length Sepal.Width Petal.Length
1            5.1         3.5          1.4
2            4.9         3.0          1.4
3            4.7         3.2          1.3
4            4.6         3.1          1.5
5            5.0         3.6          1.4
6            5.4         3.9          1.7
7            4.6         3.4          1.4
8            5.0         3.4          1.5
9            4.4         2.9          1.4
10           4.9         3.1          1.5
11           5.4         3.7          1.5
12           4.8         3.4          1.6
13           4.8         3.0          1.4
14           4.3         3.0          1.1
15           5.8         4.0          1.2
16           5.7         4.4          1.5
17           5.4         3.9          1.3
18           5.1         3.5          1.4
19           5.7         3.8          1.7
20           5.1         3.8          1.5
21           5.4         3.4          1.7
22           5.1         3.7          1.5
```

```r
(tbl_iris = as_tibble(iris))
```

```
# A tibble: 150 × 5
   Sepal.Length Sepal.Width Petal.Length
          <dbl>       <dbl>        <dbl>
 1          5.1         3.5          1.4
 2          4.9         3            1.4
 3          4.7         3.2          1.3
 4          4.6         3.1          1.5
 5          5           3.6          1.4
 6          5.4         3.9          1.7
 7          4.6         3.4          1.4
 8          5           3.4          1.5
 9          4.4         2.9          1.4
10          4.9         3.1          1.5
# ℹ 140 more rows
# ℹ 2 more variables: Petal.Width <dbl>,
#   Species <fct>
```

Tibbles are lazy (preserving type)

By default, subsetting tibbles always results in another tibble (`$` or `[[` can still be used to subset for a specific column). i.e. tibble subsets are always preserving and therefore type consistent.

```
1 tbl_iris[1,]
```

```
# A tibble: 1 × 5
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
         <dbl>       <dbl>        <dbl>       <dbl> <fct>
1          5.1         3.5          1.4         0.2 setosa
```

```
1 tbl_iris[,1]
```

```
# A tibble: 150 × 1
   Sepal.Length
          <dbl>
 1          5.1
 2          4.9
 3          4.7
 4          4.6
 5          5
 6          5.4
 7          4.6
 8          5
 9          4.4
10          4.9
# ℹ 140 more rows
```

```
1 head(tbl_iris[[1]])
```

```
[1] 5.1 4.9 4.7 4.6 5.0 5.4
```

```
1 head(tbl_iris$Species)
```

```
[1] setosa setosa setosa setosa setosa set
Levels: setosa versicolor virginica
```

Tibbles are lazy (partial matching)

Tibbles do not use partial matching when the `$` operator is used.

```
1 head( iris$Species )
```

```
[1] setosa setosa setosa setosa
Levels: setosa versicolor virgir
```

```
1 head( tbl_iris$Species )
```

```
[1] setosa setosa setosa setosa
Levels: setosa versicolor virgir
```

```
1 head( iris$Sp )
```

```
[1] setosa setosa setosa setosa
Levels: setosa versicolor virgir
```

```
1 head( tbl_iris$Sp )
```

```
Warning: Unknown or uninitialise
NULL
```

Tibbles are lazy (length coercion)

Only vectors with length 1 will undergo length coercion / recycling - anything else throws an error.

```
1 data.frame(x = 1:4, y = 1)
```

```
  x y
1 1 1
2 2 1
3 3 1
4 4 1
```

```
1 tibble(x = 1:4, y = 1)
```

```
# A tibble: 4 × 2
      x     y
  <int> <dbl>
1     1     1
2     2     1
3     3     1
4     4     1
```

```
1 data.frame(x = 1:4, y = 1:2)
```

```
  x y
1 1 1
2 2 2
3 3 1
4 4 2
```

```
1 tibble(x = 1:4, y = 1:2)
```

```
Error in `tibble()`:
! Tibble columns must have compatib
• Size 4: Existing data.
• Size 2: Column `y`.
ℹ Only values of size one are recyc
```

Tibbles and S3

```
t = tibble(
  x = 1:3,
  y = c("A","B","C")
)

class(t)
```

```
[1] "tbl_df"     "tbl"        "data.frame"
```

```
d = data.frame(
  x = 1:3,
  y = c("A","B","C")
)

class(d)
```

```
[1] "data.frame"
```

```
methods(class="tbl_df")
```

```
 [1] [             [[            [[<-          [<-           $
 [6] $<-           as.data.frame coerce        initialize    names<-
[11] Ops           row.names<-   show          slotsFromS3   str
[16] tbl_sum
see '?methods' for accessing help and source code
```

```
methods(class="tbl")
```

```
 [1] [[<-        [<-         $<-         coerce      format
 [6] glimpse     initialize  Ops         print       show
[11] slotsFromS3 tbl_sum
see '?methods' for accessing help and source code
```

Tibble support?

Tibbles are just specialized data frames, and will fall back to base data frame methods when needed.

```
d = tibble(
  x = rnorm(100),
  y = 3 + x + rnorm(100, sd = 0.1)
)
```

```
lm(y~x, data = d)
```

```
Call:
lm(formula = y ~ x, data = d)

Coefficients:
(Intercept)            x
      2.986        1.011
```

Why did this work?

magrittr

What is a pipe

> In software engineering, a pipeline consists of a chain of processing elements (processes, threads, coroutines, functions, etc.), arranged so that the output of each element is the input of the next;
>
> Wikipedia - Pipeline (software)

Magrittr's pipe (`%>%`) is an infix operator that allows us to link two functions together in a way that is readable from left to right.

The three code examples below are equivalent:

```
1 f(g(x=1, y=2), n=2)
```

```
1 g(x=1, y=2) %>% f(n=2)  # Magrittr
```

```
1 g(x=1, y=2) |> f(n=2)  # Base R >= 4.1.0
```

The current version of RStudio on the departmental servers is v4.4.1 so you are welcome to use it.

Readability

All of the following are fine; it comes down to personal preference. Try to be consistent.

Nested:

```
h( g( f(x), y=1), z=1 )
```

Piped:

```
f(x) %>%
  g(y=1) %>%
  h(z=1)
```

Intermediate:

```
res = f(x)
res = g(res, y=1)
res = h(res, z=1)
```

What about other arguments?

Sometimes we want to send our results to an function argument other than first one or we want to use the previous result for multiple arguments. In these cases we can refer to the previous result using `.`.

```
1 data.frame(a = 1:3, b = 3:1) %>% lm(a~b, data=.)
```

```
Call:
lm(formula = a ~ b, data = .)

Coefficients:
(Intercept)            b
          4           -1
```

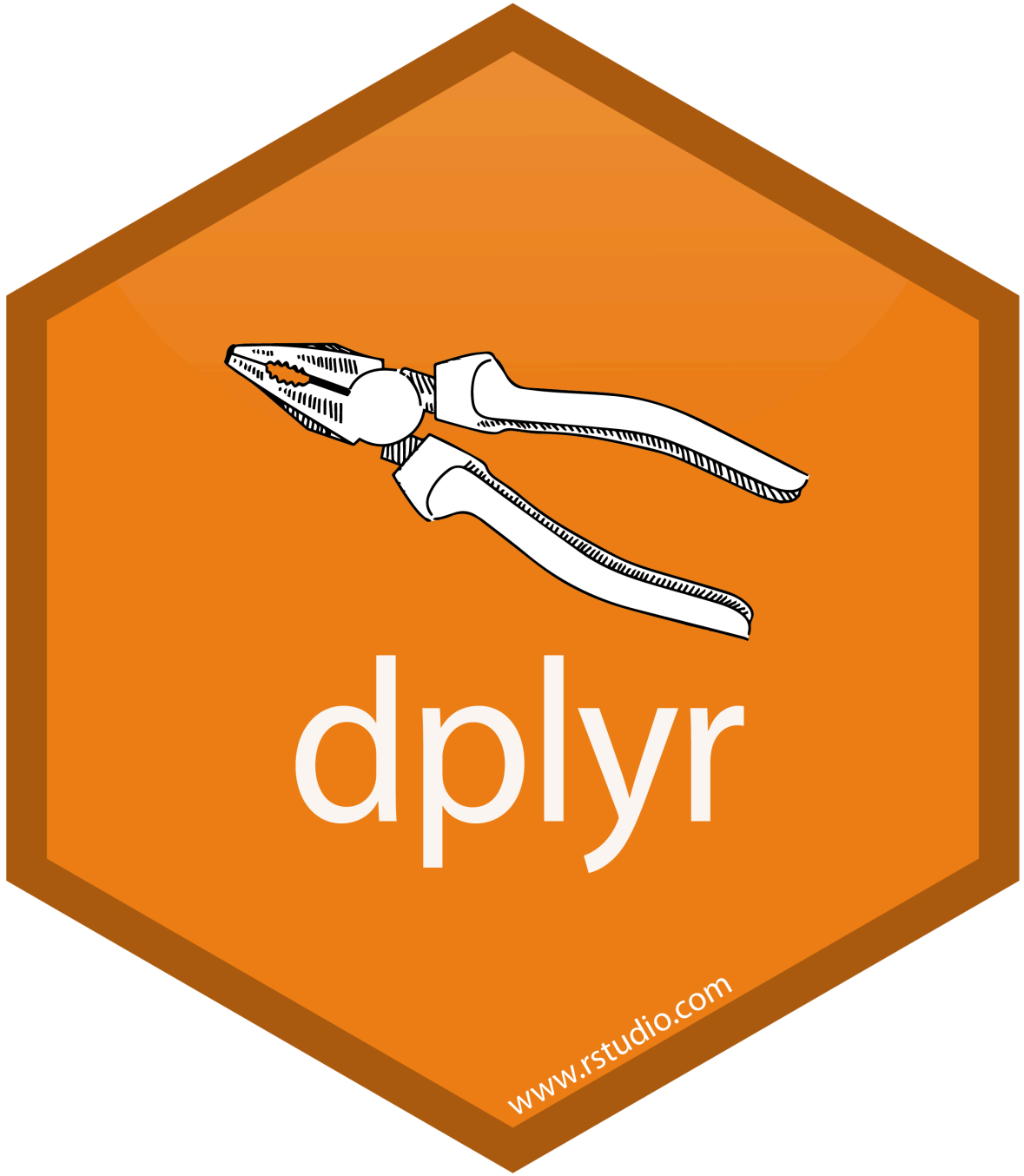
```
1 data.frame(a = 1:3, b = 3:1) %>% .[[1]]
```

```
[1] 1 2 3
```

```
1 data.frame(a = 1:3, b = 3:1) %>% .[[length(.)]]
```

```
[1] 3 2 1
```

The above example does not work with Base R pipe, for two reasons: `|>` uses `_` instead of `.` and it only allows one call of the "pipe placeholder".


dplyr
www.rstudio.com

A Grammar of Data Manipulation

dplyr is based on the concepts of functions as verbs that manipulate data frames.

Core single data frame functions / verbs:

- `filter()` / `slice()` - pick rows based on criteria
- `mutate()` / `transmute()` - create or modify columns
- `summarise()` / `count()` - reduce variables to values
- `group_by()` / `ungroup()` - modify other verbs to act on subsets
- `select()` / `rename()` - select columns by name
- `pull()` - grab a column as a vector
- `arrange()` - reorder rows
- `distinct()` - filter for unique rows
- `relocate()` - change column order
- … (many more)

dplyr rules

1. First argument is *always* a data frame
2. Subsequent arguments say what to do with the data frame
3. *Always* return a data frame
4. Don’t modify in place (how does this affect memory usage?)
5. Magic via non-standard evaluation + lazy evaluation and S3

Example Data

We will demonstrate dplyr's functionality using the nycflights13 data.

```
1 library(dplyr)
2 library(nycflights13)
3
4 flights
```

```
# A tibble: 336,776 × 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
# ℹ 336,766 more rows
```

```
# ℹ 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>
```

filter() - March flights

```
1 flights |> filter(month == 3)
```

```
# A tibble: 28,834 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013     3     1        4           2159       125      318
 2  2013     3     1       50           2358        52      526
 3  2013     3     1      117           2245       152      223
 4  2013     3     1      454            500        -6      633
 5  2013     3     1      505            515       -10      746
 6  2013     3     1      521            530        -9      813
 7  2013     3     1      537            540        -3      856
 8  2013     3     1      541            545        -4     1014
 9  2013     3     1      549            600       -11      639
10  2013     3     1      550            600       -10      747
# ℹ 28,824 more rows
# ℹ 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
```

filter() - Flights in the first 7 days of March

```
1 flights |> filter(month == 3, day <= 7)
```

```
# A tibble: 6,530 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013     3     1        4           2159       125      318
 2  2013     3     1       50           2358        52      526
 3  2013     3     1      117           2245       152      223
 4  2013     3     1      454            500        -6      633
 5  2013     3     1      505            515       -10      746
 6  2013     3     1      521            530        -9      813
 7  2013     3     1      537            540        -3      856
 8  2013     3     1      541            545        -4     1014
 9  2013     3     1      549            600       -11      639
10  2013     3     1      550            600       -10      747
# ℹ 6,520 more rows
# ℹ 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
```

filter() - Flights to LAX *or* JFK in March

```
1 flights |> filter(dest == "LAX" | dest == "JFK", month == 3)
```

```
# A tibble: 1,178 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013     3     1      607            610        -3      832
 2  2013     3     1      629            632        -3      844
 3  2013     3     1      657            700        -3      953
 4  2013     3     1      714            715        -1      939
 5  2013     3     1      716            710         6      958
 6  2013     3     1      727            730        -3     1007
 7  2013     3     1      836            840        -4     1111
 8  2013     3     1      857            900        -3     1202
 9  2013     3     1      903            900         3     1157
10  2013     3     1      904            831        33     1150
# ℹ 1,168 more rows
# ℹ 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
```

slice() - First 10 flights

```
1 flights |> slice(1:10)
```

```
# A tibble: 10 × 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
# ℹ 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, des
```

slice_tail() - Last 3 flights

```
1 flights |> slice_tail(n = 3)
```

```
# A tibble: 3 × 19
   year month   day dep_time sched_dep_time dep_delay arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013     9    30       NA           1210        NA       NA
2  2013     9    30       NA           1159        NA       NA
3  2013     9    30       NA            840        NA       NA
# ℹ 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, des
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>
```

Could also do

```
1 flights |> slice((n()-2):n()) # n() returns number of rows
```

select() - Individual Columns

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

```
# A tibble: 336,776 × 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
# ℹ 336,766 more rows
```

select() - Exclude Columns

```
1 flights |> select(-year, -month, -day)
```

```
# A tibble: 336,776 × 16
   dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
      <int>          <int>     <dbl>    <int>          <int>     <dbl>
 1      517            515         2      830            819        11
 2      533            529         4      850            830        20
 3      542            540         2      923            850        33
 4      544            545        -1     1004           1022       -18
 5      554            600        -6      812            837       -25
 6      554            558        -4      740            728        12
 7      555            600        -5      913            854        19
 8      557            600        -3      709            723       -14
 9      557            600        -3      838            846        -8
10      558            600        -2      753            745         8
# ℹ 336,766 more rows
# ℹ 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>
```

select() - Ranges

```
1 flights |> select(year:day)  # Also supports exclusion with -
```

```
# A tibble: 336,776 × 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
# ℹ 336,766 more rows
```

select() - Matching with `tidyselect`

```
1 flights |> select(contains("dep"), starts_with("arr"))
```

```
# A tibble: 336,776 × 5
   dep_time sched_dep_time dep_delay arr_time arr_delay
      <int>          <int>     <dbl>    <int>     <dbl>
 1      517            515         2      830        11
 2      533            529         4      850        20
 3      542            540         2      923        33
 4      544            545        -1     1004       -18
 5      554            600        -6      812       -25
 6      554            558        -4      740        12
 7      555            600        -5      913        19
 8      557            600        -3      709       -14
 9      557            600        -3      838        -8
10      558            600        -2      753         8
# ℹ 336,766 more rows
```

Other helpers provide by tidyselect:

select() + where() - Get numeric columns

```
1 flights |> select(where(is.numeric))  # Also supports exclusion wit
```

```
# A tibble: 336,776 × 14
    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
# ℹ 336,766 more rows
# ℹ 7 more variables: sched_arr_time <int>, arr_delay <dbl>, flight <int
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>
```

rename() - Change column names

```
flights |>
  relocate(tailnum) |>  # Relocate column to the left
  rename(tail_number = tailnum)
```

```
# A tibble: 336,776 × 19
   tail_number  year month   day dep_time sched_dep_time dep_delay arr_time
   <chr>       <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1 N14228       2013     1     1      517            515         2      830
 2 N24211       2013     1     1      533            529         4      850
 3 N619AA       2013     1     1      542            540         2      923
 4 N804JB       2013     1     1      544            545        -1     1004
 5 N668DN       2013     1     1      554            600        -6      812
 6 N39463       2013     1     1      554            558        -4      740
 7 N516JB       2013     1     1      555            600        -5      913
 8 N829AS       2013     1     1      557            600        -3      709
 9 N593JB       2013     1     1      557            600        -3      838
10 N3ALAA       2013     1     1      558            600        -2      753
# ℹ 336,766 more rows
# ℹ 11 more variables: sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

pull()

```
1 names(flights)
```

```
 [1] "year"           "month"          "day"            "dep_time"
 [5] "sched_dep_time" "dep_delay"      "arr_time"       "sched_arr_time"
 [9] "arr_delay"      "carrier"        "flight"         "tailnum"
[13] "origin"         "dest"           "air_time"       "distance"
[17] "hour"           "minute"         "time_hour"
```

```
1 flights |> pull("year") |> head()
```

```
[1] 2013 2013 2013 2013 2013 2013
```

```
1 flights |> pull(1) |> head()
```

```
[1] 2013 2013 2013 2013 2013 2013
```

```
1 flights |> pull(-1) |> head()
```

```
[1] "2013-01-01 05:00:00 EST" "2013-01-01 05:00:00 EST"
[3] "2013-01-01 05:00:00 EST" "2013-01-01 05:00:00 EST"
[5] "2013-01-01 06:00:00 EST" "2013-01-01 05:00:00 EST"
```

arrange() - Sort data

```
1 flights |>
2   mutate(original_row_number = row_number()) |>
3   relocate(original_row_number, origin, dest) |>
4   filter(month == 3,day == 2) |>
5   arrange(origin, dest) # Use arrange(desc(origin), dest) for descending order
```

```
# A tibble: 765 × 20
   original_row_number origin dest   year month   day dep_time
                 <int> <chr>  <chr> <int> <int> <int>    <int>
 1              137570 EWR    ALB    2013     3     2     1336
 2              137257 EWR    ATL    2013     3     2      628
 3              137262 EWR    ATL    2013     3     2      637
 4              137303 EWR    ATL    2013     3     2      743
 5              137391 EWR    ATL    2013     3     2      857
 6              137455 EWR    ATL    2013     3     2     1027
 7              137491 EWR    ATL    2013     3     2     1134
 8              137592 EWR    ATL    2013     3     2     1412
 9              137726 EWR    ATL    2013     3     2     1633
10              137750 EWR    ATL    2013     3     2     1655
# ℹ 755 more rows
# ℹ 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>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

distinct() - Find unique rows

```
1 flights |>
2   select(origin, dest) |>
3   distinct() |>  # Use distinct(.keep_all = TRUE) to keep all the columns
4   arrange(origin,dest)
```

```
# A tibble: 224 × 2
   origin dest
   <chr>  <chr>
 1 EWR    ALB
 2 EWR    ANC
 3 EWR    ATL
 4 EWR    AUS
 5 EWR    AVL
 6 EWR    BDL
 7 EWR    BNA
 8 EWR    BOS
 9 EWR    BQN
10 EWR    BTV
# ℹ 214 more rows
```

mutate() - Modify / create columns

```
1 library(lubridate)  # Date manipulation library
2
3 flights |>
4   select(year:day) |>
5   mutate(date = paste(year, month, day, sep="-"),
6          days_until_2025 = as.numeric(ymd("2025-01-01") - ymd(date)))
```

```
# A tibble: 336,776 × 5
    year month   day date     days_until_2025
   <int> <int> <int> <chr>              <dbl>
 1  2013     1     1 2013-1-1            4383
 2  2013     1     1 2013-1-1            4383
 3  2013     1     1 2013-1-1            4383
 4  2013     1     1 2013-1-1            4383
 5  2013     1     1 2013-1-1            4383
 6  2013     1     1 2013-1-1            4383
 7  2013     1     1 2013-1-1            4383
 8  2013     1     1 2013-1-1            4383
 9  2013     1     1 2013-1-1            4383
10  2013     1     1 2013-1-1            4383
# ℹ 336,766 more rows
```

summarise() - Arregate rows

```
flights |>
  summarize(
    n = n(),
    min_dep_delay = min(dep_delay, na.rm = TRUE),
    max_dep_delay = max(dep_delay, na.rm = TRUE)
  )
```

```
# A tibble: 1 × 3
      n min_dep_delay max_dep_delay
  <int>         <dbl>         <dbl>
1 336776          -43          1301
```

group_by()

```
1 flights |> group_by(origin)
```

```
# A tibble: 336,776 × 19
# Groups:   origin [3]
    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
# ℹ 336,766 more rows
# ℹ 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>
```

summarise() with group_by()

```
1 flights |>
2   group_by(origin) |>
3   summarize(
4     n = n(),
5     min_dep_delay = min(dep_delay, na.rm = TRUE),
6     max_dep_delay = max(dep_delay, na.rm = TRUE)
7   )
```

```
# A tibble: 3 × 4
  origin      n min_dep_delay max_dep_delay
  <chr>   <int>         <dbl>         <dbl>
1 EWR    120835           -25          1126
2 JFK    111279           -43          1301
3 LGA    104662           -33           911
```

Groups after summarise

```
1 flights |>
2   group_by(origin, month) |>
3   summarize(
4     n = n(),
5     min_dep_delay = min(dep_delay, na.rm = TRUE),
6     max_dep_delay = max(dep_delay, na.rm = TRUE)
7   )
```

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

```
# A tibble: 36 × 5
# Groups:   origin [3]
   origin month     n min_dep_delay max_dep_delay
   <chr>  <int> <int>         <dbl>         <dbl>
 1 EWR        1  9893           -21          1126
 2 EWR        2  9107           -21           786
 3 EWR        3 10420           -22           443
 4 EWR        4 10531           -21           545
 5 EWR        5 10592           -20           878
 6 EWR        6 10175           -19           502
 7 EWR        7 10475           -18           653
 8 EWR        8 10359           -17           424
 9 EWR        9  9550           -23           486
```

Avoid the message

```
flights |>
  group_by(origin, month) |>
  summarize(
    n = n(),
    min_dep_delay = min(dep_delay, na.rm = 
    max_dep_delay = max(dep_delay, na.rm = 
    .groups = "drop"
  )
```

```
# A tibble: 36 × 5
   origin month     n min_dep_delay
   <chr>  <int> <int>         <dbl>
 1 EWR        1  9893           -21
 2 EWR        2  9107           -21
 3 EWR        3 10420           -22
 4 EWR        4 10531           -21
 5 EWR        5 10592           -20
 6 EWR        6 10175           -19
 7 EWR        7 10475           -18
 8 EWR        8 10359           -17
 9 EWR        9  9550           -23
10 EWR       10 10104           -25
# ℹ 26 more rows
# ℹ 1 more variable: max_dep_delay <dbl>
```

```
flights |>
  group_by(origin, month) |>
  summarize(
    n = n(),
    min_dep_delay = min(dep_delay, na.rm = 
    max_dep_delay = max(dep_delay, na.rm = 
    .groups = "keep"
  )
```

```
# A tibble: 36 × 5
# Groups:   origin, month [36]
   origin month     n min_dep_delay
   <chr>  <int> <int>         <dbl>
 1 EWR        1  9893           -21
 2 EWR        2  9107           -21
 3 EWR        3 10420           -22
 4 EWR        4 10531           -21
 5 EWR        5 10592           -20
 6 EWR        6 10175           -19
 7 EWR        7 10475           -18
 8 EWR        8 10359           -17
 9 EWR        9  9550           -23
10 EWR       10 10104           -25
# ℹ 26 more rows
# ℹ 1 more variable: max_dep_delay <dbl>
```

The .by argument

The `.by` (and `by`) arguments are used for per operation grouping while `group_by()` is intended for persistent grouping. See `?dplyr_by` for more details and examples.

```
flights |>
  summarize(
    n = n(),
    min_dep_delay = min(dep_delay, na.rm=TRUE),
    max_dep_delay = max(dep_delay, na.rm=TRUE),
    .by = origin
  )
```

```
# A tibble: 3 × 4
  origin      n min_dep_delay max_dep_delay
  <chr>   <int>         <dbl>         <dbl>
1 EWR    120835           -25          1126
2 LGA    104662           -33           911
3 JFK    111279           -43          1301
```

count()

```
flights |>
  summarize(
    n = n(),
    .by = c(origin, carrier)
  )
```

```
# A tibble: 35 × 3
   origin carrier     n
   <chr>  <chr>   <int>
 1 EWR    UA      46087
 2 LGA    UA       8044
 3 JFK    AA      13783
 4 JFK    B6      42076
 5 LGA    DL      23067
 6 EWR    B6       6557
 7 LGA    EV       8826
 8 LGA    AA      15459
 9 JFK    UA       4534
10 LGA    B6       6002
# ℹ 25 more rows
```

```
flights |>
  count(origin, carrier)
```

```
# A tibble: 35 × 3
   origin carrier     n
   <chr>  <chr>   <int>
 1 EWR    9E       1268
 2 EWR    AA       3487
 3 EWR    AS        714
 4 EWR    B6       6557
 5 EWR    DL       4342
 6 EWR    EV      43939
 7 EWR    MQ       2276
 8 EWR    OO          6
 9 EWR    UA      46087
10 EWR    US       4405
# ℹ 25 more rows
```

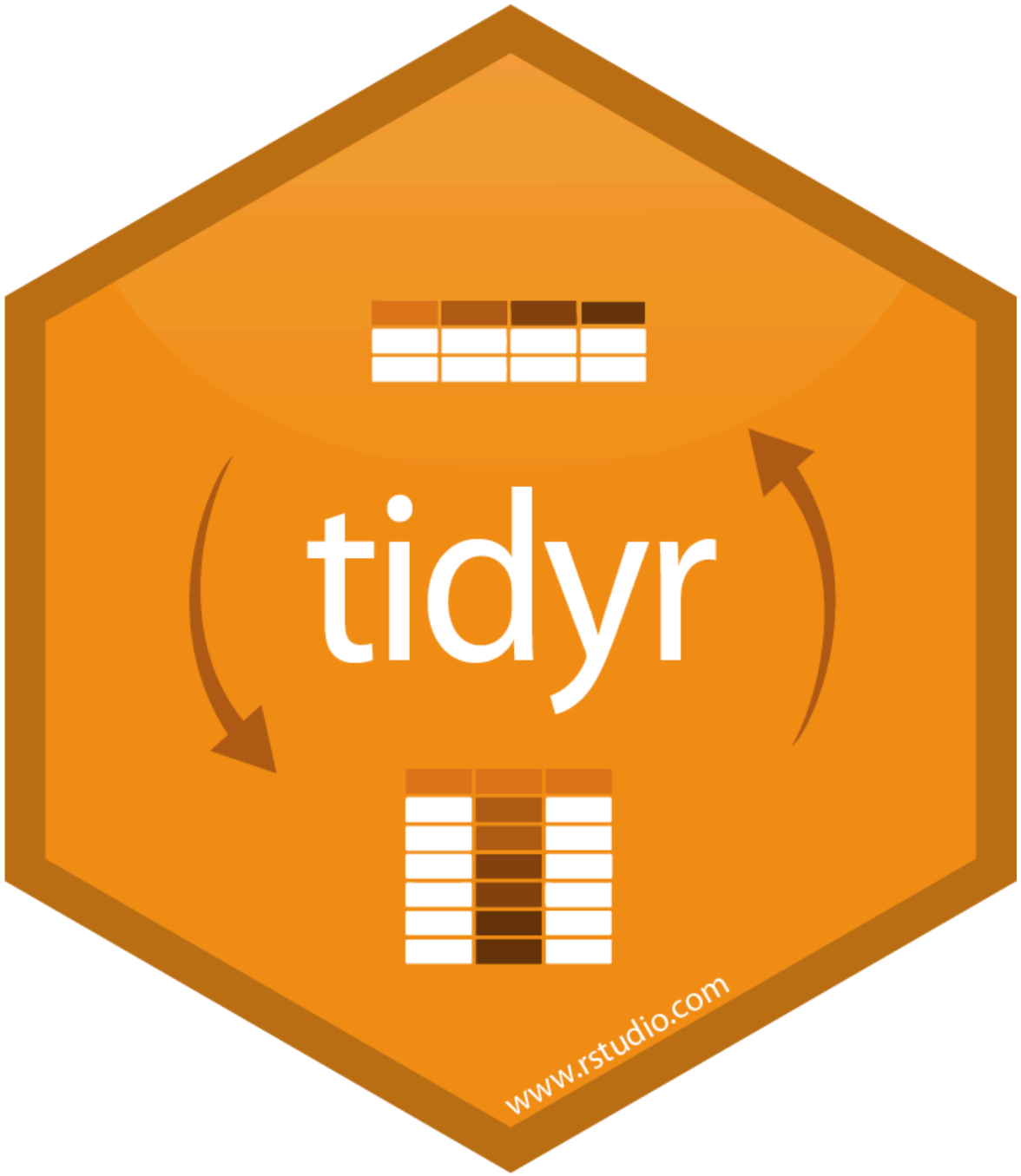
mutate() with .by

```
1 flights |>
2   mutate(mean_ad = mean(arr_delay, na.rm = T),
3          sd_ad = sd(arr_delay, na.rm = T), .by = origin) |>
4   mutate(arrival_delay_normalized_by_origin = (arr_delay - me
5   select(origin, arr_delay, last_col():last_col(2))
```

```
# A tibble: 336,776 × 5
   origin arr_delay arrival_delay_normalized_by_origin sd_ad mean
   <chr>      <dbl>                               <dbl> <dbl>   <d
 1 EWR           11                              0.0416  45.5    9
 2 LGA           20                              0.324   43.9    4
 3 JFK           33                              0.620   44.3    4
 4 JFK          -18                             -0.532   44.3    4
 5 LGA          -25                             -0.702   43.9    4
 6 EWR           12                              0.0635  45.5    9
 7 EWR           19                              0.217   45.5    9
 8 LGA          -14                             -0.451   43.9    4
 9 JFK           -8                             -0.306   44.3    4
```

Exercises / Examples

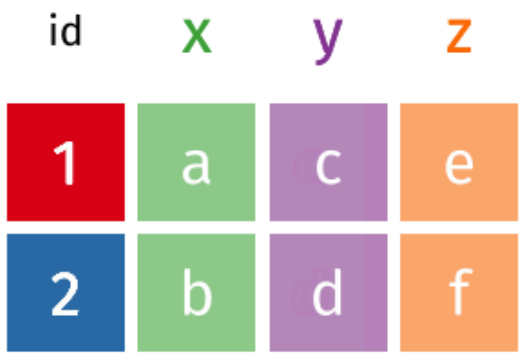
1. How many flights to Los Angeles (LAX) did each of the legacy carriers (AA, UA, DL or US) have in May from JFK, and what was their average duration?
2. What was the shortest flight out of each airport in terms of distance? In terms of duration?
3. Which plane (check the tail number) flew out of each New York airport the most?
4. Which date should you fly on if you want to have the lowest possible average departure delay? What about arrival delay?


tidyr
www.rstudio.com

Reshaping data

Wide vs Long

wide



Wide -> Long

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

`pivot_longer` (previously `gather` or `reshape2::melt`)

Syntax

```
(d = tibble::tribble(
  ~country, ~"1999",  ~"2000",
        "A", "0.7K",    "2K",
        "B",  "37K",   "80K",
        "C", "212K",  "213K"
))
```

```
# A tibble: 3 × 3
  country `1999` `2000`
  <chr>   <chr>  <chr>
1 A       0.7K   2K
2 B       37K    80K
3 C       212K   213K
```

```
pivot_longer(
  d,
  cols = "1999":"2000",
  names_to = "year",
  values_to = "cases"
)
```

```
# A tibble: 6 × 3
  country year  cases
  <chr>   <chr> <chr>
1 A       1999  0.7K
2 A       2000  2K
3 B       1999  37K
4 B       2000  80K
5 C       1999  212K
6 C       2000  213K
```

Long -> Wide

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

`pivot_wider` (previously `spread`)

Syntax

```
( d = tibble::tribble(
    ~country, ~year,   ~type, ~count,
         "A",  1999, "cases", "0.7K",
         "A",  1999,   "pop",  "19M",
         "A",  2000, "cases",   "2K",
         "A",  2000,   "pop",  "20M",
         "B",  1999, "cases",  "37K",
         "B",  1999,   "pop", "172M",
         "B",  2000, "cases", " 80K",
         "B",  2000,   "pop", "174M",
         "C",  1999, "cases", "212K",
         "C",  1999,   "pop",   "1T",
         "C",  2000, "cases", "213K",
         "C",  2000,   "pop",   "1T"
  )
)
```

```
# A tibble: 12 × 4
   country  year type  count
   <chr>   <dbl> <chr> <chr>
 1 A        1999 cases "0.7K"
 2 A        1999 pop   "19M"
 3 A        2000 cases "2K"
 4 A        2000 pop   "20M"
 5 B        1999 cases "37K"
 6 B        1999 pop   "172M"
 7 B        2000 cases " 80K"
 8 B        2000 pop   "174M"
 9 C        1999 cases "212K"
10 C        1999 pop   "1T"
11 C        2000 cases "213K"
12 C        2000 pop   "1T"
```

```
pivot_wider(
  d,
  id_cols = country:year,
  names_from = type,
  values_from = count
)
```

```
# A tibble: 6 × 4
  country  year cases  pop
  <chr>   <dbl> <chr>  <chr>
1 A        1999 "0.7K" 19M
2 A        2000 "2K"   20M
3 B        1999 "37K"  172M
4 B        2000 " 80K" 174M
5 C        1999 "212K" 1T
6 C        2000 "213K" 1T
```

Exercise 1

The `palmerpenguin` package contains measurement data on various penguin species on islands near Palmer Station in Antarctica. The code below shows the # of each species measured on each of the three islands (missing island, penguin pairs implies that species does not occur on that island).

```
palmerpenguins::penguins |>
  count(island, species)
```

```
# A tibble: 5 × 3
  island    species       n
  <fct>     <fct>     <int>
1 Biscoe    Adelie       44
2 Biscoe    Gentoo      124
3 Dream     Adelie       56
4 Dream     Chinstrap    68
5 Torgersen Adelie       52
```

Starting from these data construct a contingency table of counts for island (rows) by species (columns) using the pivot functions we've just discussed.

05:00

Separate - wider

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172
B	2000	80K	174

```
1 separate_wider_delim(d, rate, delim = "/", names = c("cases", "pop"))
```

```
# A tibble: 6 × 4
  country  year cases pop
  <chr>   <dbl> <chr> <chr>
1 A        1999 0.7K  19M
2 A        2000 2K    20M
3 B        1999 37K   172M
4 B        2000 80K   174M
5 C        1999 212K  1T
6 C        2000 213K  1T
```

Separate - longer

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M

country	year	rate
A	1999	0.7K
A	1999	19M
A	2000	2K
A	2000	20M
B	1999	37K
B	1999	172M
B	2000	80K
B	2000	174M

```
1 separate_longer_delim(d, rate, delim =
```

```
# A tibble: 12 × 3
   country  year rate
   <chr>   <dbl> <chr>
 1 A        1999 0.7K
 2 A        1999 19M
 3 A        2000 2K
 4 A        2000 20M
 5 B        1999 37K
 6 B        1999 172M
 7 B        2000 80K
 8 B        2000 174M
 9 C        1999 212K
10 C        1999 1T
11 C        2000 213K
12 C        2000 1T
```

Other separates

In previous versions of tidyr there was a single catch-all `separate()` function. This still exists and is available in the package but it is **superseded**.

Other helpful separate functions:

- `separate_longer_position()`
- `separate_wider_position()`
- `separate_wider_regex()`

Unite

country	century	year
Afghan	19	99
Afghan	20	0
Brazil	19	99
Brazil	20	0
China	19	99
China	20	0

country	year
Afghan	1999
Afghan	2000
Brazil	1999
Brazil	2000
China	1999
China	2000

```
1 unite(d, century, year, col = "year", sep = "")
```

```
# A tibble: 6 × 2
  country year
  <chr>   <chr>
1 Afghan  1999
2 Afghan  2000
3 Brazil  1999
4 Brazil  2000
5 China   1999
6 China   2000
```

Example 1 - tidy grades

Is the following data tidy?

```
grades = tibble::tribble(
  ~name,   ~hw_1, ~hw_2, ~hw_3, ~hw_4, ~proj_1, ~proj_2,
  "Alice",    19,    19,    18,    20,      89,      95,
  "Bob",      18,    20,    18,    16,      77,      88,
  "Carol",    18,    20,    18,    17,      96,      99,
  "Dave",     19,    19,    18,    19,      86,      82
)
```

How would we calculate a final score based on the following formula,

$$\text{score} = 0.5 \frac{\sum_i \text{hw}_i}{80} + 0.5 \frac{\sum_j \text{proj}_j}{200}$$

Semi-tidy approach

```
grades |>
  mutate(
    hw_avg = (hw_1+hw_2+hw_3+hw_4)/4,
    proj_avg = (proj_1+proj_2)/2
  ) |>
  mutate(
    overall = 0.5*(proj_avg/100) + 0.5*(hw_avg/20)
  )
```

```
# A tibble: 4 × 10
  name   hw_1  hw_2  hw_3  hw_4 proj_1 proj_2 hw_avg proj_avg overall
  <chr> <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl>  <dbl>    <dbl>   <dbl>
1 Alice    19    19    18    20     89     95   19        92     0.935
2 Bob      18    20    18    16     77     88   18        82.5   0.862
3 Carol    18    20    18    17     96     99   18.2      97.5   0.944
4 Dave     19    19    18    19     86     82   18.8      84     0.889
```

pivot_longer (Wide -> Long)

```
tidyr::pivot_longer(
  grades,
  cols = hw_1:proj_2,
  names_to = "assignment",
  values_to = "score"
)
```

```
# A tibble: 24 × 3
   name  assignment score
   <chr> <chr>      <dbl>
 1 Alice hw_1          19
 2 Alice hw_2          19
 3 Alice hw_3          18
 4 Alice hw_4          20
 5 Alice proj_1        89
 6 Alice proj_2        95
 7 Bob   hw_1          18
 8 Bob   hw_2          20
 9 Bob   hw_3          18
10 Bob   hw_4          16
# ℹ 14 more rows
```

Split type and id

```
tidyr::pivot_longer(
  grades,
  cols = hw_1:proj_2,
  names_to = c("type", "id"),
  names_sep = "_",
  values_to = "score"
)
```

```
# A tibble: 24 × 4
   name  type  id    score
   <chr> <chr> <chr> <dbl>
 1 Alice hw    1        19
 2 Alice hw    2        19
 3 Alice hw    3        18
 4 Alice hw    4        20
 5 Alice proj  1        89
 6 Alice proj  2        95
 7 Bob   hw    1        18
 8 Bob   hw    2        20
 9 Bob   hw    3        18
10 Bob   hw    4        16
# ℹ 14 more rows
```

Tidy approach?

```
grades |>
  tidyr::pivot_longer(
    cols = hw_1:proj_2,
    names_to = c("type", "id"),
    names_sep = "_",
    values_to = "score"
  ) |>
  summarize(
    total = sum(score),
    .by = c(name, type)
  )
```

```
# A tibble: 8 × 3
  name  type  total
  <chr> <chr> <dbl>
1 Alice hw       76
2 Alice proj    184
3 Bob   hw       72
4 Bob   proj    165
5 Carol hw       73
6 Carol proj    195
7 Dave  hw       75
8 Dave  proj    168
```

pivot_wider - (Long -> Wide)

```
grades |>
  tidyr::pivot_longer(
    cols = hw_1:proj_2,
    names_to = c("type", "id"),
    names_sep = "_",
    values_to = "score"
  ) |>
  summarize(
    total = sum(score),
    .by = c(name, type)
  ) |>
  tidyr::pivot_wider(
    names_from = type,
    values_from = total
  )
```

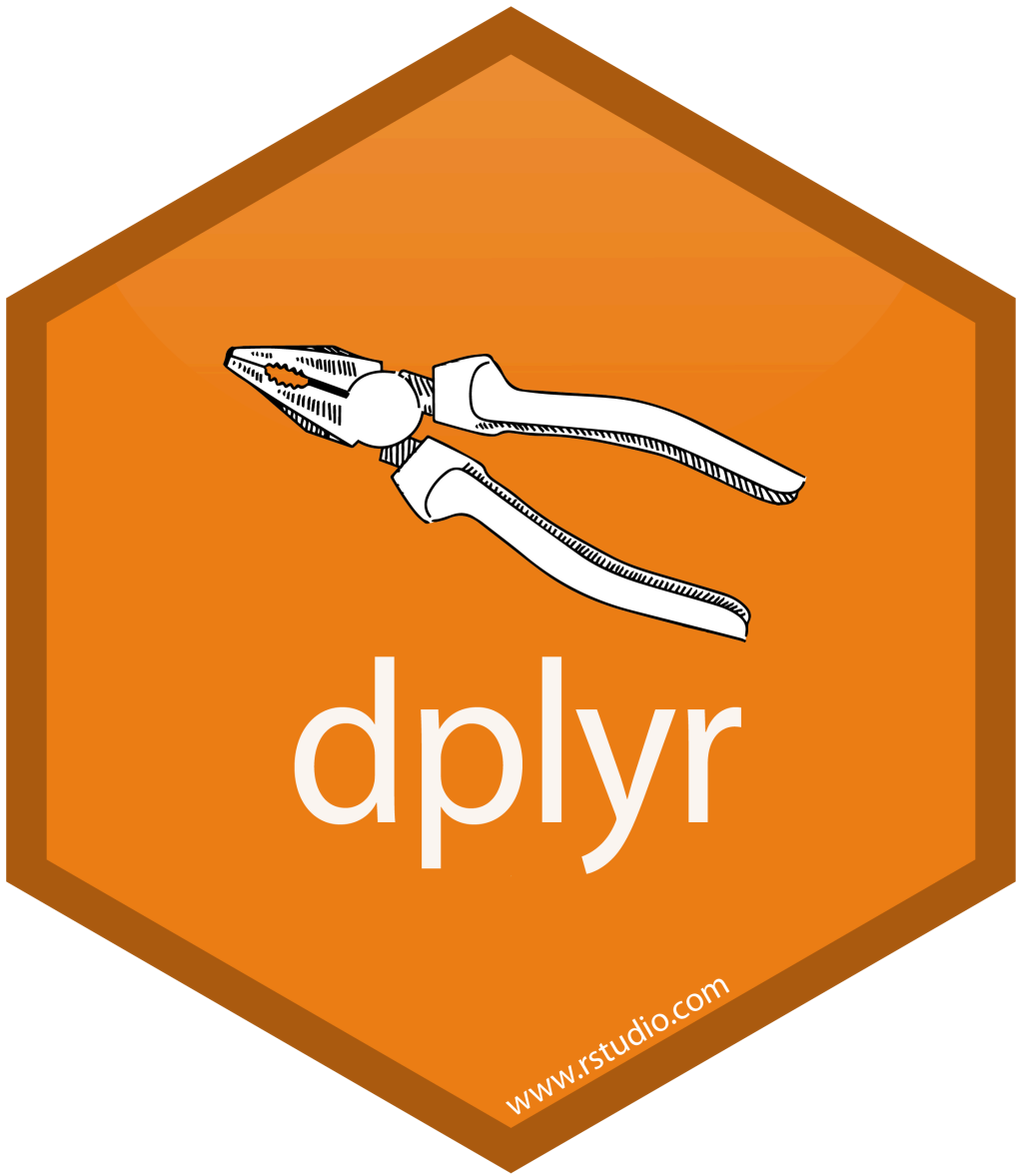
```
# A tibble: 4 × 3
  name     hw  proj
  <chr> <dbl> <dbl>
1 Alice    76   184
2 Bob      72   165
3 Carol    73   195
4 Dave     75   168
```

Wrapping up

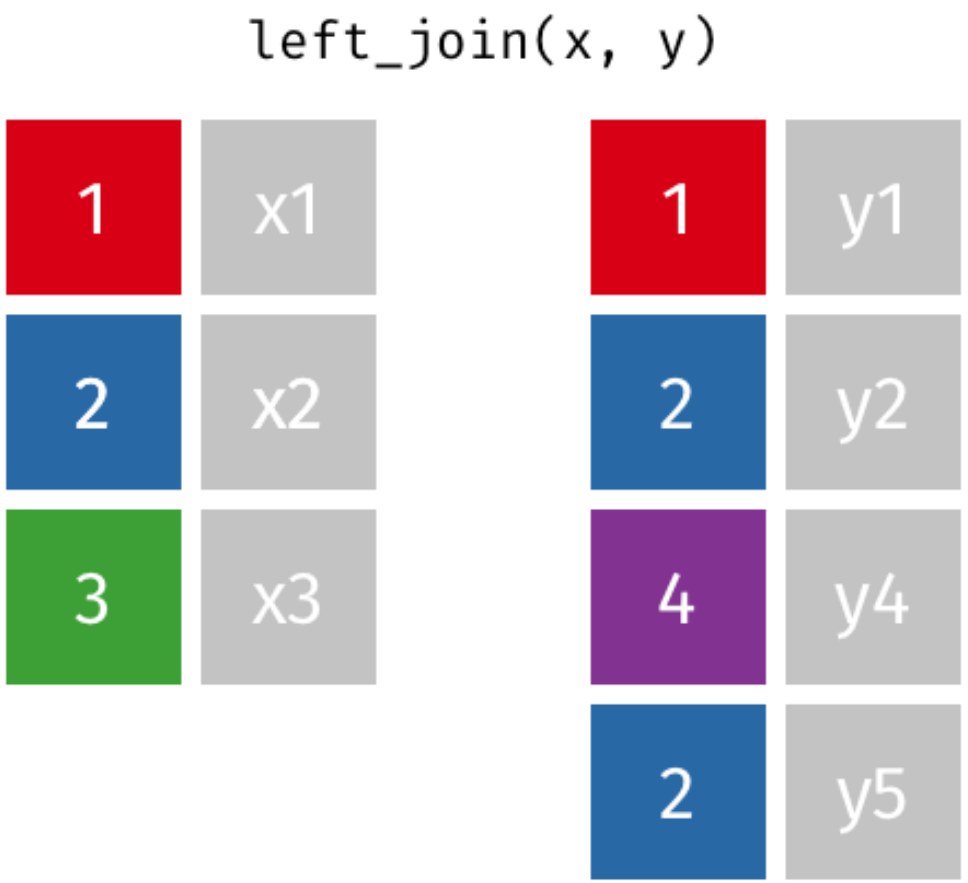
```
final_grades <- grades |>
  tidyr::pivot_longer(
    cols = hw_1:proj_2,
    names_to = c("type", "id"),
    names_sep = "_",
    values_to = "score"
  ) |>
  summarize(
    total = sum(score),
    .by = c(name, type)
  ) |>
  tidyr::pivot_wider(
    names_from = type,
    values_from = total
  ) |>
  mutate(
    score = 0.5*(hw/80) +
            0.5*(proj/200)
  )

```

```
# A tibble: 4 × 4
  name     hw  proj score
  <chr> <dbl> <dbl> <dbl>
1 Alice    76   184 0.935
2 Bob      72   165 0.862
3 Carol    73   195 0.944
4 Dave     75   168 0.889
```

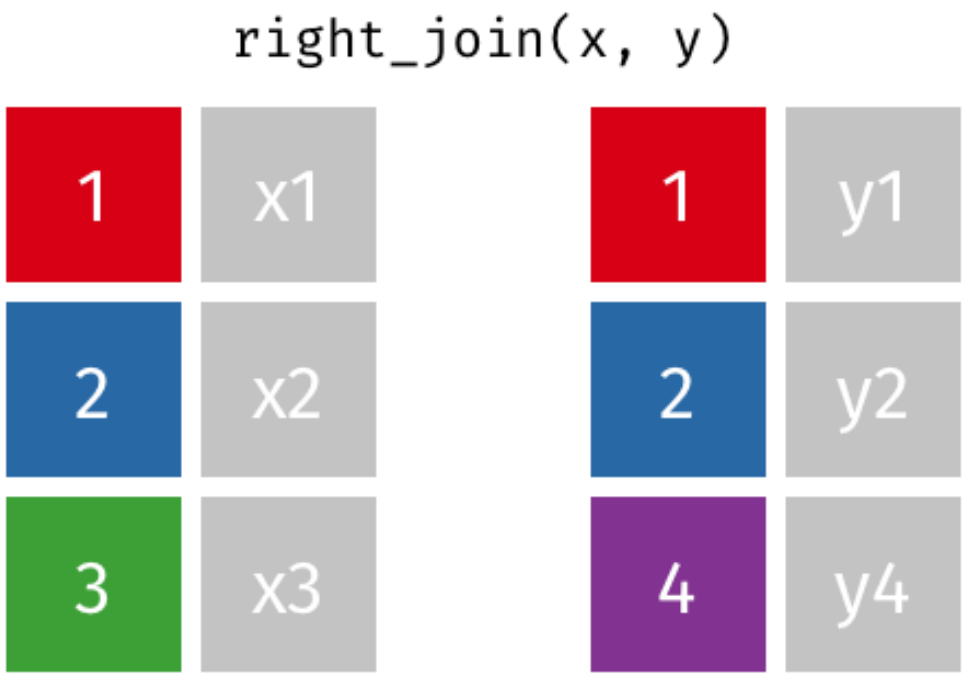
Joins



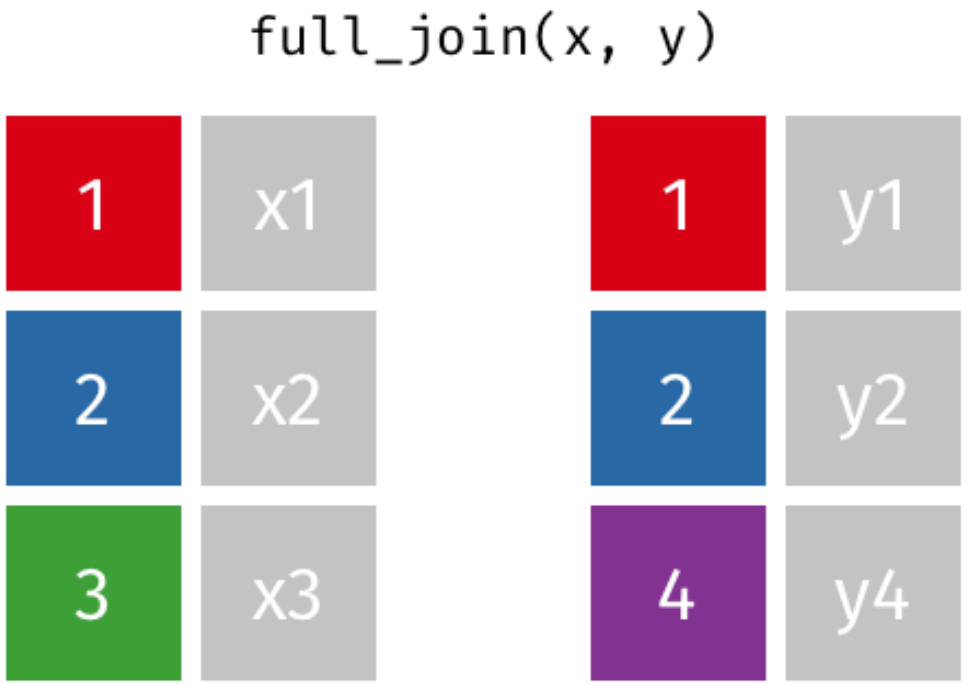
Joins (left)



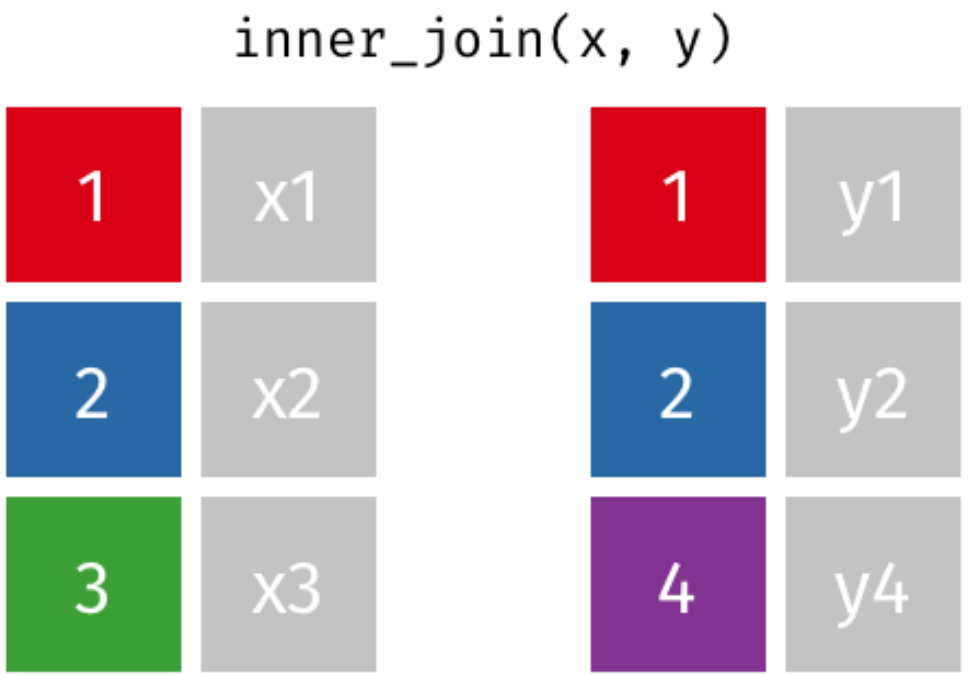
Joins (right)



Joins (full / outer)



Joins (inner)



join by

By default dplyr's join functions will join based on intersecting column names between the two data frames.

To specify the columns to join by (or to handle non-matching names) pass in a character vector of column names (or a named character vector where the names match the left data frame and the values match the right).

Recently more advanced joins have been implemented using the `join_by()` construct which allows for: equality, inequality, rolling, overlap, and cross joins. See `?join_by` for details.

Back to School!

```
behavior_reports <- tibble::tribble(
  ~incident_name,   ~person_1,   ~person_2,     ~person_3,
  "Food fight",     "alice,bob", "carole,dave", NA,
  "'Kick me'",      "alice",     "carole",      NA,
  "Flash mob",      "bob",       "carol",       "Dave",
  "Playing games",  "Alice",     "Bob",         NA
) |>
  pivot_longer(person_1:person_3,
    values_to = "person_involved",
    values_drop_na = TRUE) |>
  separate_longer_delim(person_involved, delim = ",") |>
  summarise(num_incidents = n(), .by = person_involved)
behavior_reports
```

```
# A tibble: 8 × 2
  person_involved num_incidents
  <chr>                   <int>
1 alice                       2
2 bob                         2
3 carole                      2
4 dave                        1
5 carol                       1
6 Dave                        1
7 Alice                       1
8 Bob                         1
```

Left/Right Join

Inner/Full Joins

A possible solution!

```
1 library(fuzzyjoin)  # Allows us to join by nearest string
2
3 fuzzy_joined <- stringdist_inner_join(final_grades,
4   behavior_reports,
5   by = c("name" = "person_involved"))
6
7 # After the join we still need to do a little tidying, what's left?
8 fuzzy_joined
```

```
# A tibble: 8 × 6
  name     hw  proj score person_involved num_incidents
  <chr> <dbl> <dbl> <dbl> <chr>                   <int>
1 Alice    76   184 0.935 alice                       2
2 Alice    76   184 0.935 Alice                       1
3 Bob      72   165 0.862 bob                         2
4 Bob      72   165 0.862 Bob                         1
5 Carol    73   195 0.944 carole                      2
6 Carol    73   195 0.944 carol                       1
7 Dave     75   168 0.889 dave                        1
8 Dave     75   168 0.889 Dave                        1
```

This is like using a chainsaw to break a twig. How could we do this in a more principled manner?