COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

GR5206: lecture 7

Computational Statistics
And Introduction to Data Science

Thibault Vatter

Department of Statistics, Columbia University

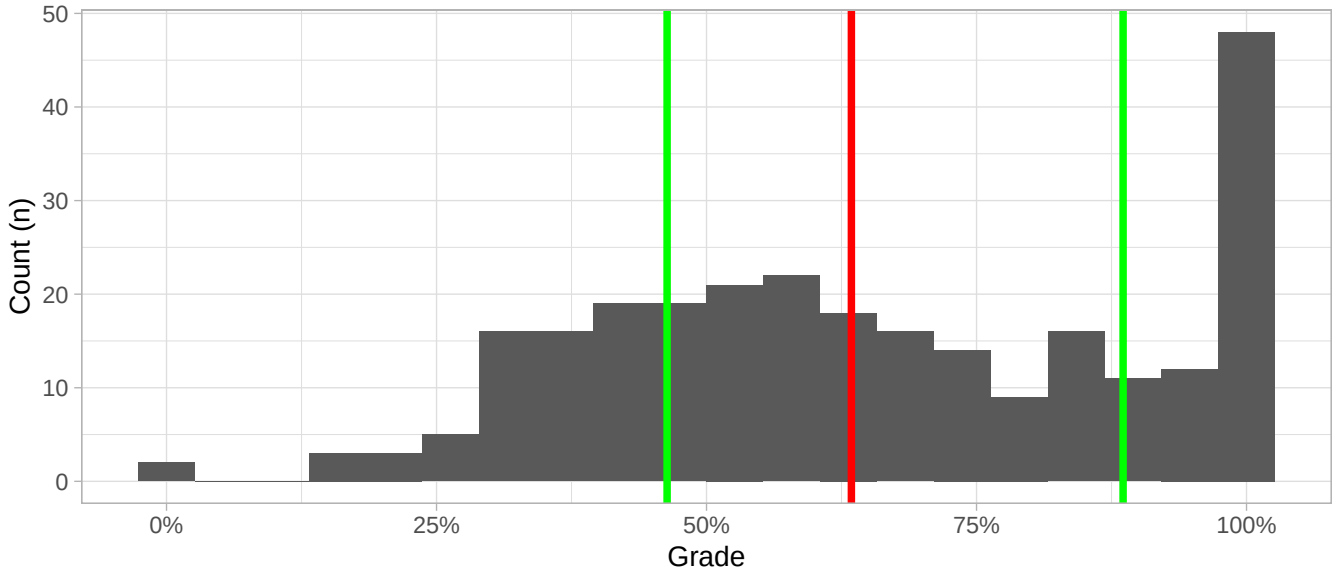
10/25/2019

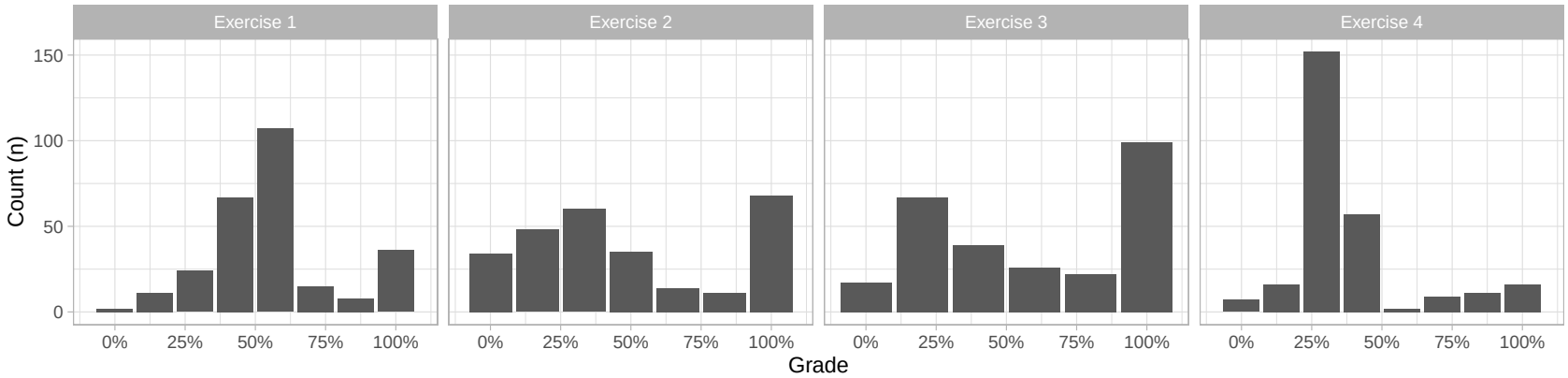
The midterm

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

Midterm Grade Distribution:

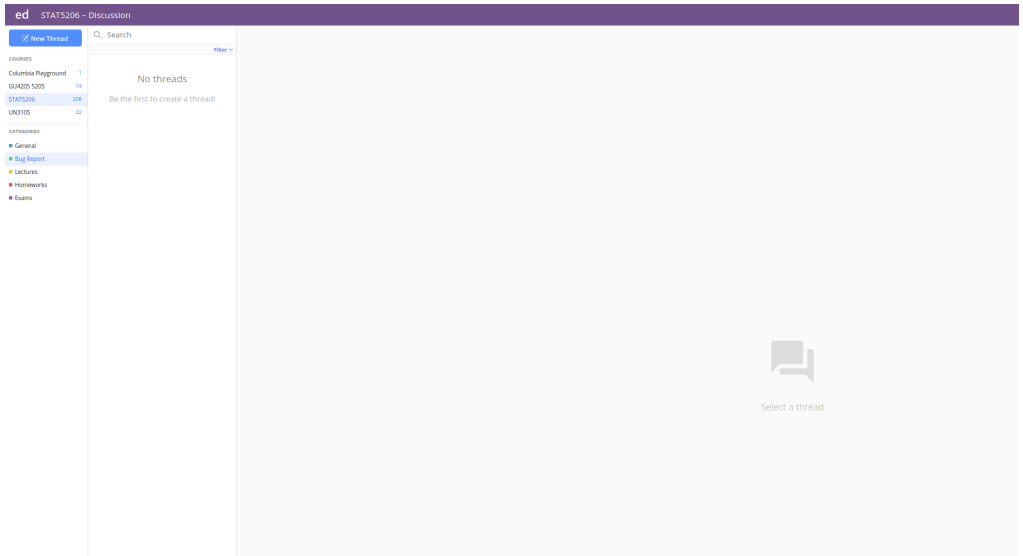
The median is in red, the 25th and 75th percentiles are in green.





Stability of Ed

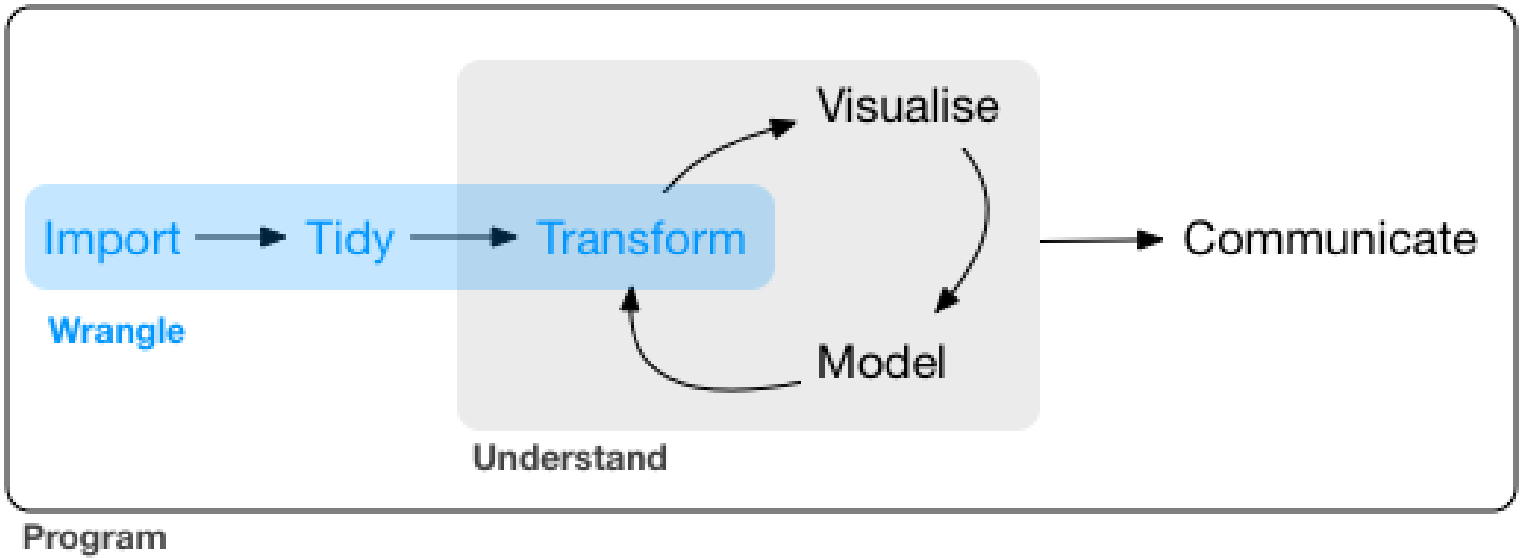
- Two important updates this week solving:
  - Most RStudio crashes.
  - Memory overloads.
- A new discussion category to file bug reports:
  - Ed people will follow this closely!
  - Post there when you have an issue with:
    - Platform stability (e.g., crashes).
    - Missing features (e.g., toggle to fullscreen in SEB)



The final

- Regarding interaction between Ed/SEB and your laptop:
  - Test in advance!
  - For instance, use SEB to complete parts of your HW...
- Regarding the room connectivity issues:
  - Either better rooms on Friday, December 8th.
    - **Most likely in the evening, save the date!**
    - **Discussing with colleagues will get you an automatic F!**
  - Or paper exam (MCQ).
- Regarding the course's grade:
  - I will increase of the weight of the final (over that of the midterm) if you improve!

Today



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

R for data science

Outline

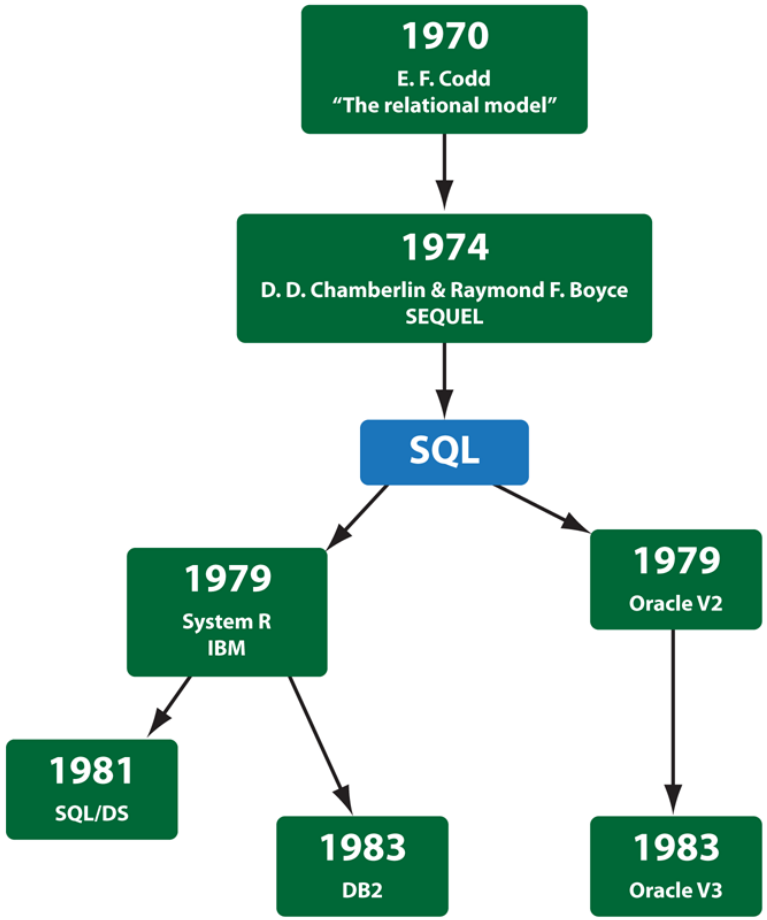
Outline

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


```
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


```
airports
#> # A tibble: 1,458 x 8
#>    faa   name                  lat    lon   alt    tz dst   tzone
#>    <chr> <chr>               <dbl>  <dbl> <int> <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
```

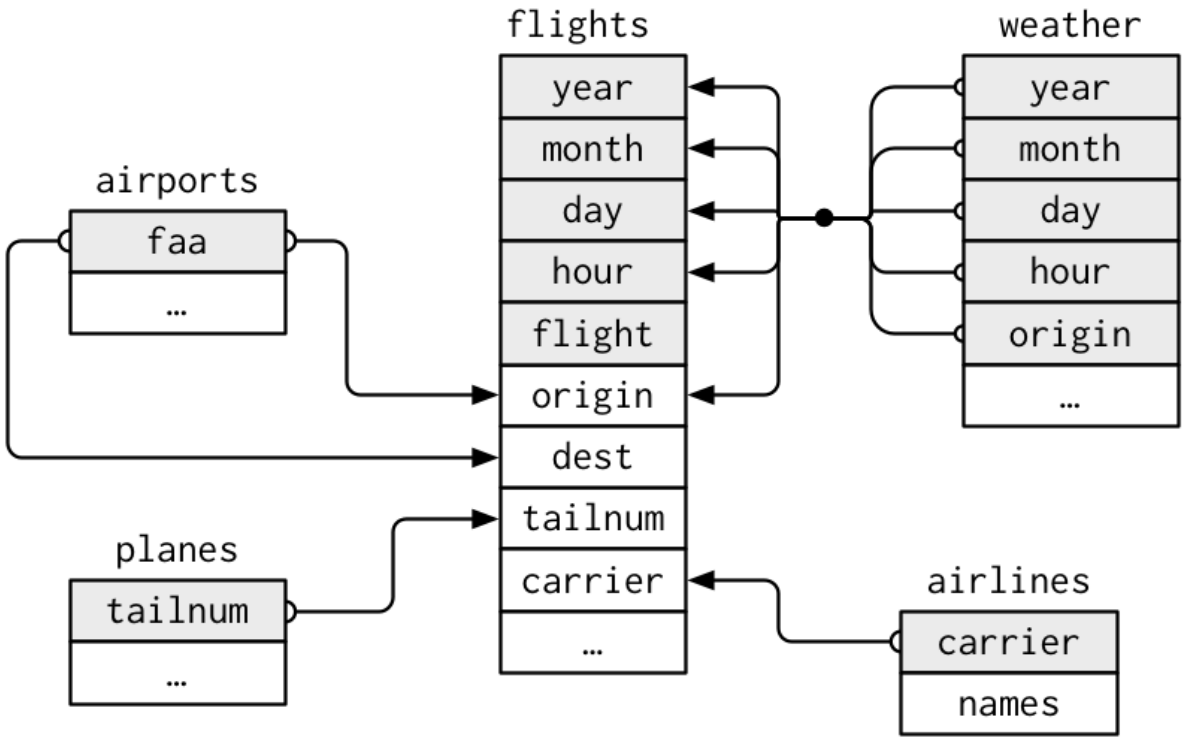
nycflights13::planes


```
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
```

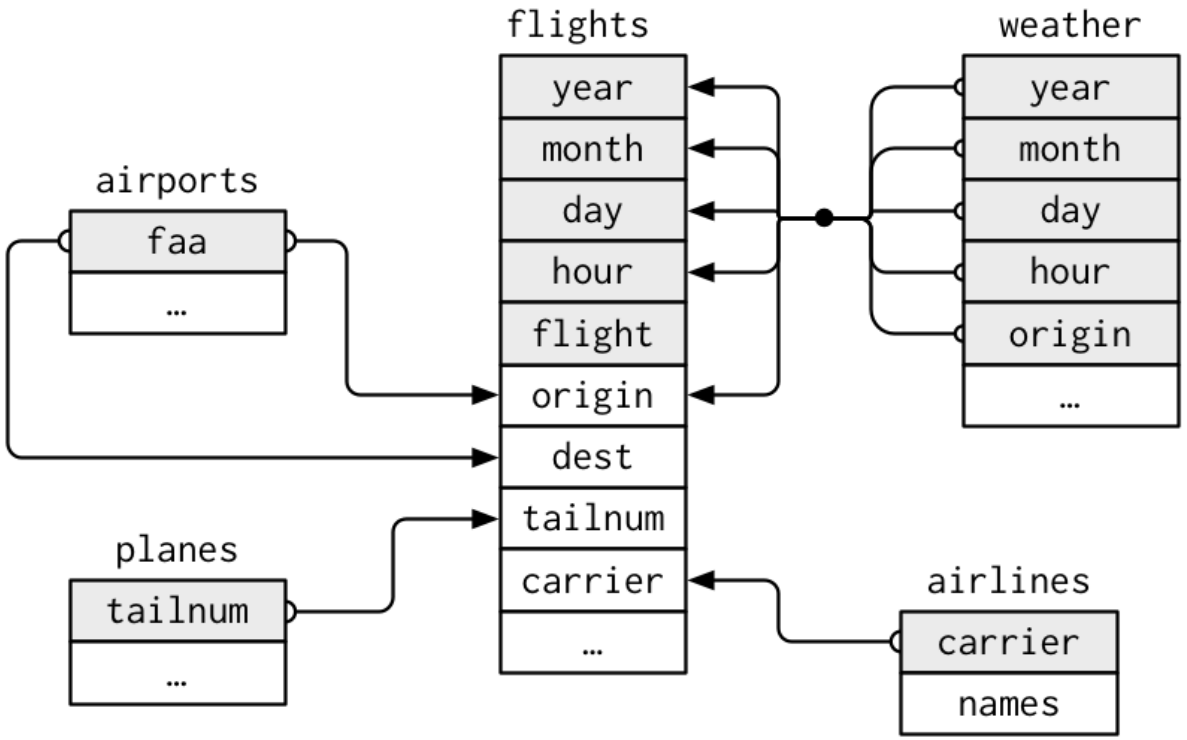
nycflights13::weather


```
weather
#> # A tibble: 26,115 x 15
#>    origin  year month   day  hour  temp  dewp humid wind_dir
#>    <chr>  <dbl> <dbl> <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

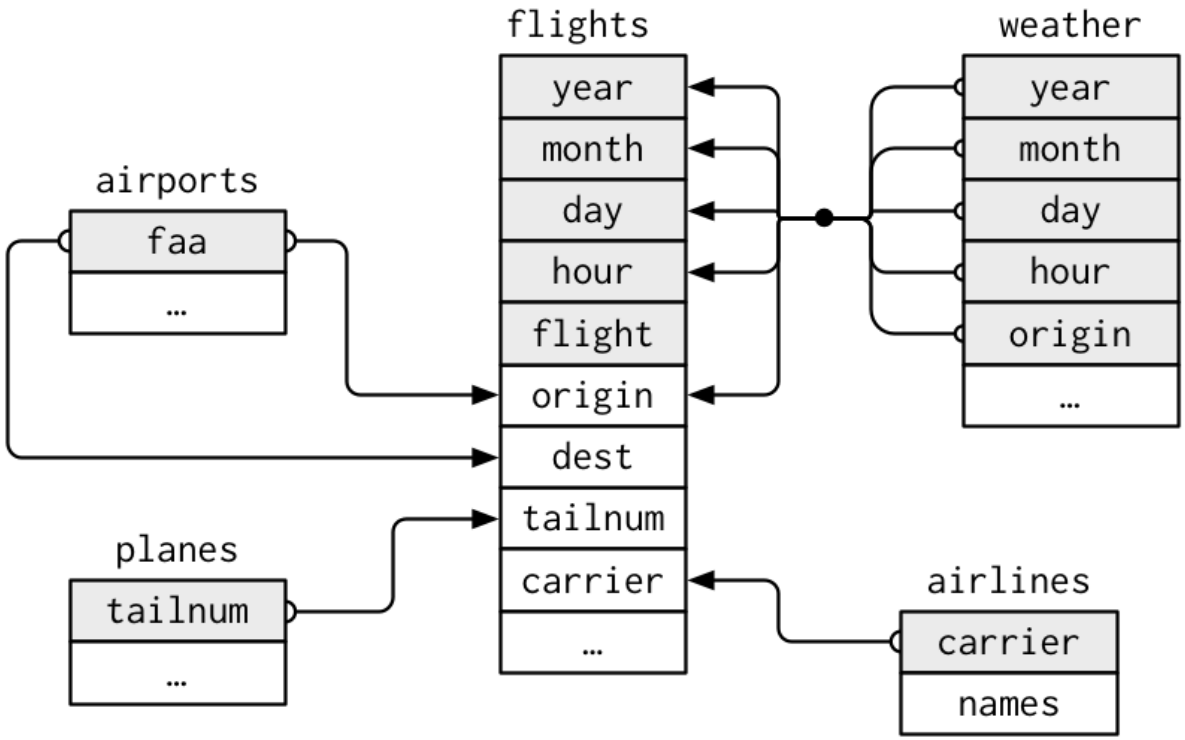


Exercise 1



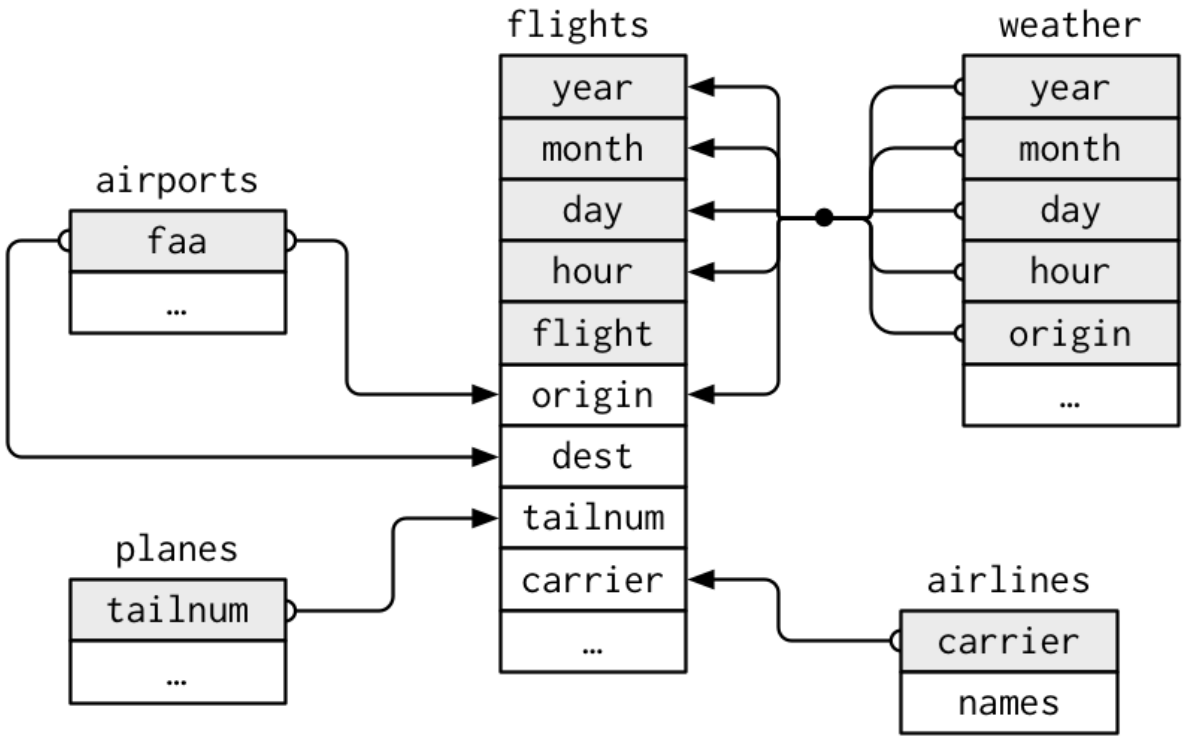
- Imagine you wanted to draw (approximately) the route each plane flies from its origin to its destination.
  - What variables would you need?
  - What tables would you need to combine?

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?


```
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
#>   <dbl> <dbl> <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?


```
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**.

Outline

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


```
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

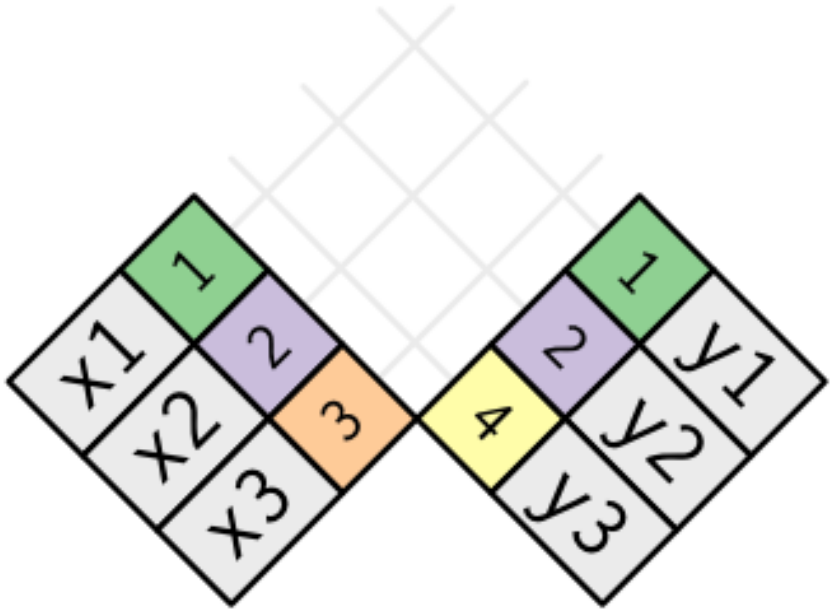
```
flights2 %>%
  select(-origin, -dest) %>%
  left_join(airlines, by = "carrier")
#> # A tibble: 336,776 x 7
#>     year month   day  hour tailnum carrier name
#>    <int> <int> <int> <dbl> <chr>   <chr>   <chr>
#>  1  2013     1     1     5 N14228  UA      United Air Lines Inc.
#>  2  2013     1 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

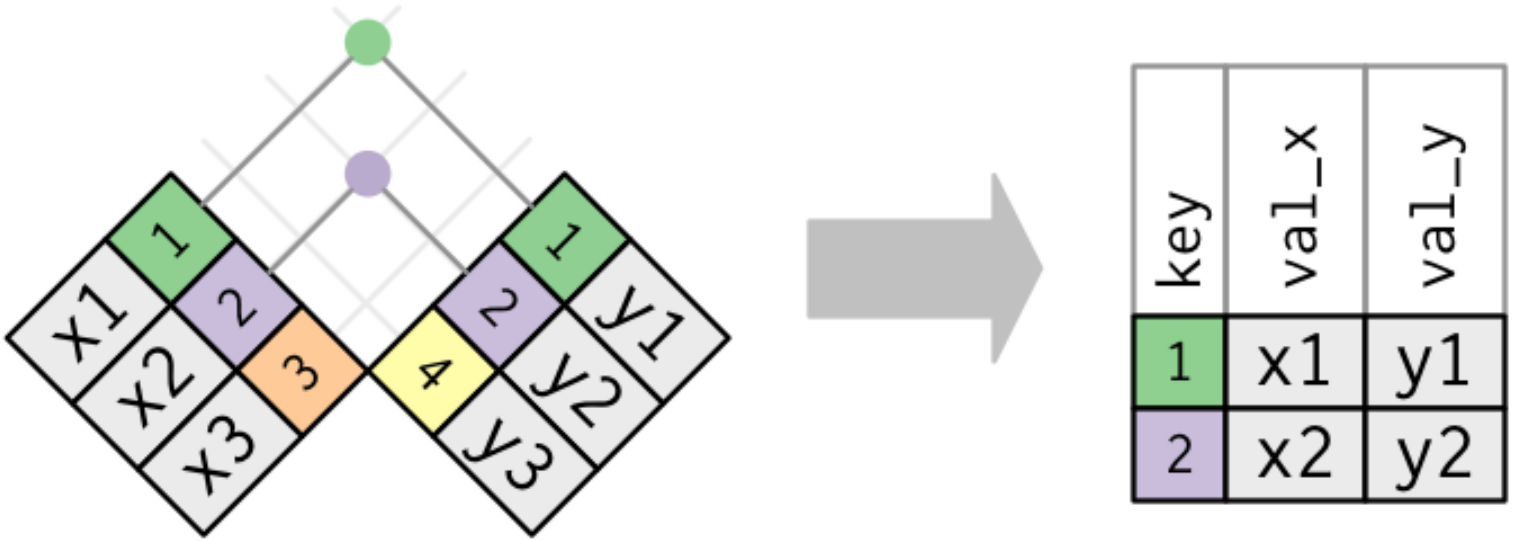

```
flights2 %>%
  select(-origin, -dest) %>%
  mutate(name = airlines$name[match(carrier, airlines$carrier)])
#> # A tibble: 336,776 x 7
#>     year month   day  hour tailnum carrier name
#>    <int> <int> <int> <dbl> <chr>   <chr>   <chr>
#>  1  2013     1     1     5 N14228  UA      United Air Lines 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


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



Inner join

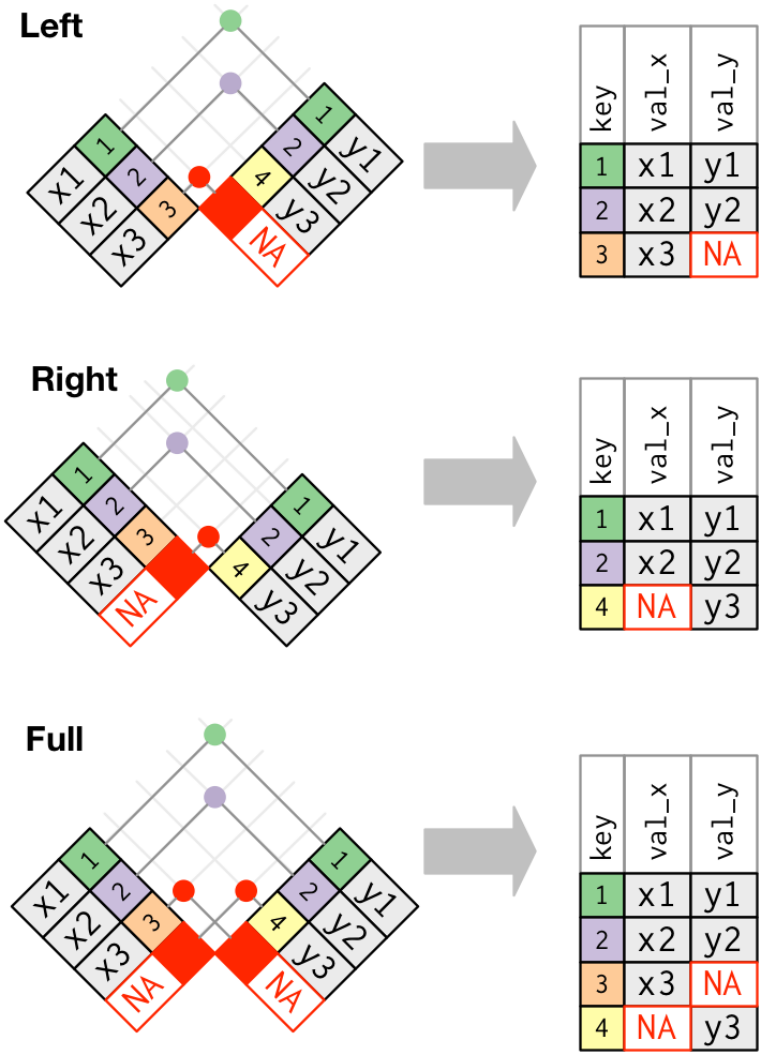



```
x %>%
  inner_join(y, by = "key")
#> # A tibble: 2 x 3
#>     key val_x val_y
#>   <dbl> <chr> <chr>
#> 1     1 x1    y1
#> 2     2 x2    y2
```

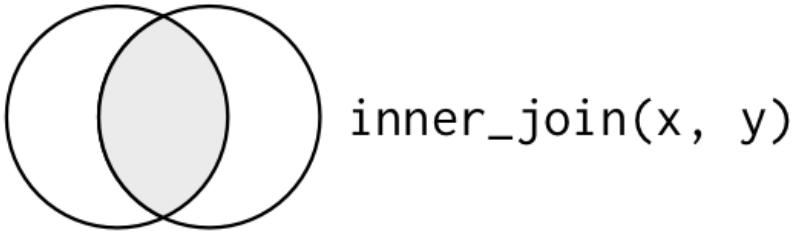
Outer joins

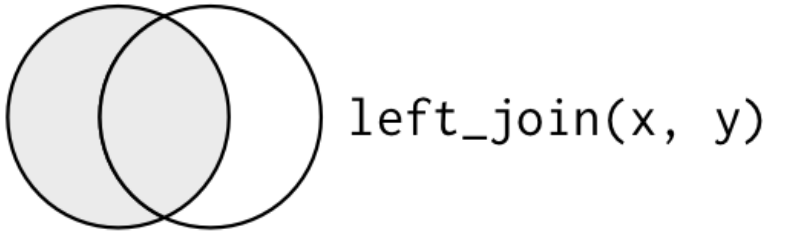
- **Outer joins** keep observations that appear in at least one of the tables:
  - **Left join:** keeps all observations in `x`.
  - **Right join:** keeps all observations in `y`.
  - **Full join:** keeps all observations in `x` and `y`
- They work by adding to each table an additional "virtual" observation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

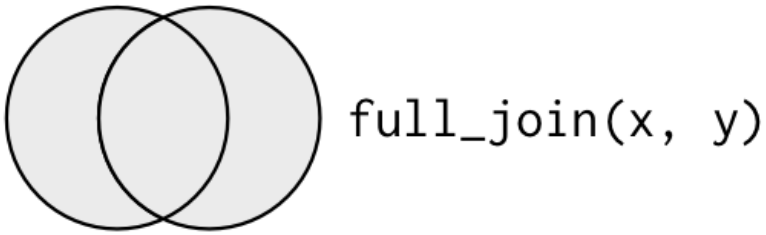
Outer joins II

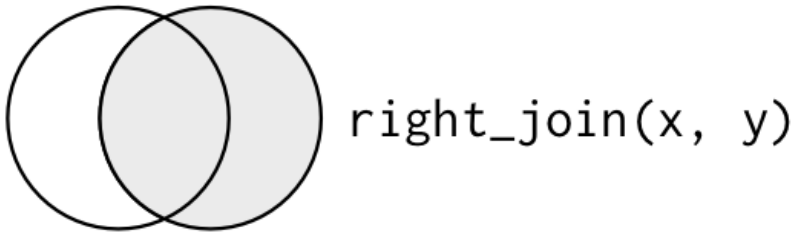


A Venn diagram for joins









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

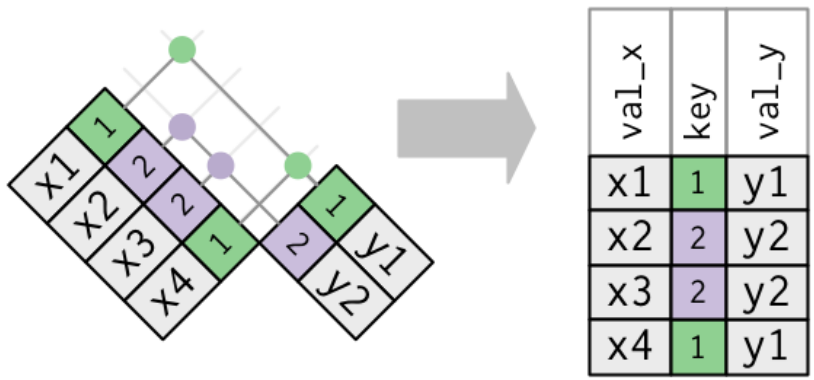
- Only `x` has duplicated keys:

```
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
```



Both tables have duplicate keys

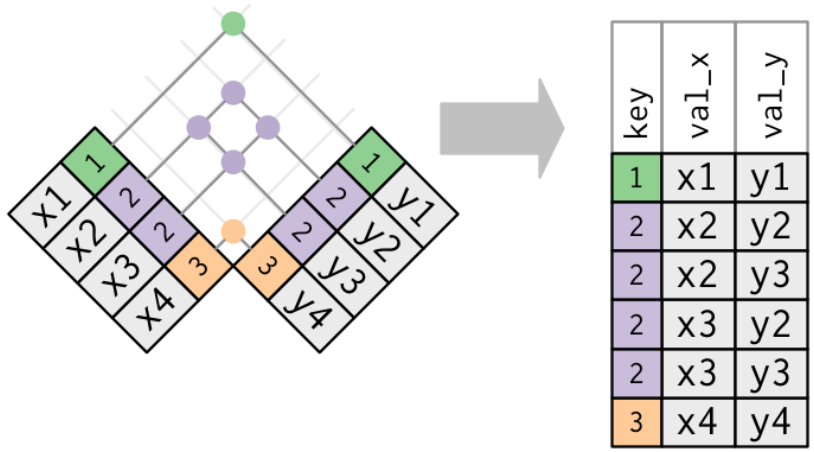
- Both x and y have duplicated keys:

```
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
#>    <dbl> <dbl> <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

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- 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      Geor~  30.0
#>  2  2013     1     1     5 LGA    IAH   N24211  UA      Geor~  30.0
#>  3  2013     1     1     5 JFK    MIA   N619AA  AA      Miam~  25.8
#>  4  2013     1     1     5 JFK    BQN   N804JB  B6      <NA>   NA
#>  5  2013     1     1     6 LGA    ATL   N668DN  DL      Hart~  33.6
#>  6  2013     1     1     5 EWR    ORD   N39463  UA      Chic~  42.0
#>  7  2013     1     1     6 EWR    FLL   N516JB  B6      Fort~  26.1
#>  8  2013     1     1     6 LGA    IAD   N829AS  EV      Wash~  38.9
#>  9  2013     1     1     6 JFK    MCO   N593JB  B6      Orla~  28.4
#> 10  2013     1     1     6 LGA    ORD   N3ALAA  AA      Chic~  42.0
#> # ... with 336,766 more rows, and 5 more variables: lon <dbl>,
#> #   alt <int>, tz <dbl>, dst <chr>, tzone <chr>
```

Other implementations

- `base::merge()` can perform all four types of mutating join:

dplyr	merge
`inner_join(x, y)`	`merge(x, y)`
`left_join(x, y)`	`merge(x, y, all.x = TRUE)`
`right_join(x, y)`	`merge(x, y, all.y = TRUE),`
`full_join(x, y)`	`merge(x, y, all.x = TRUE, all.y = TRUE)`

- Advantages of the specific dplyr verbs:
  - More clearly convey the intent of your code.
  - Considerably faster and don't mess with the order of the rows.
  - Conversion to SQL using `dbplyr`.

Other implementations II

- SQL is the inspiration for dplyr's conventions:

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`.

Filtering 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 1.411e+05 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?

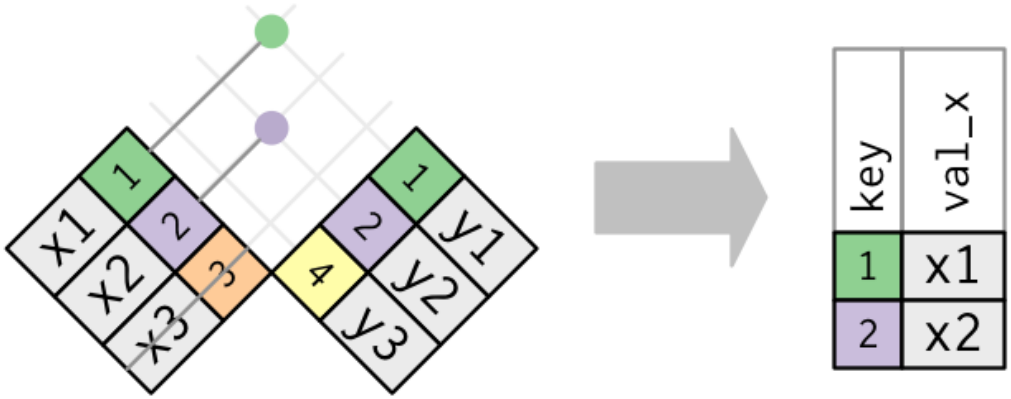
Semi-join

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

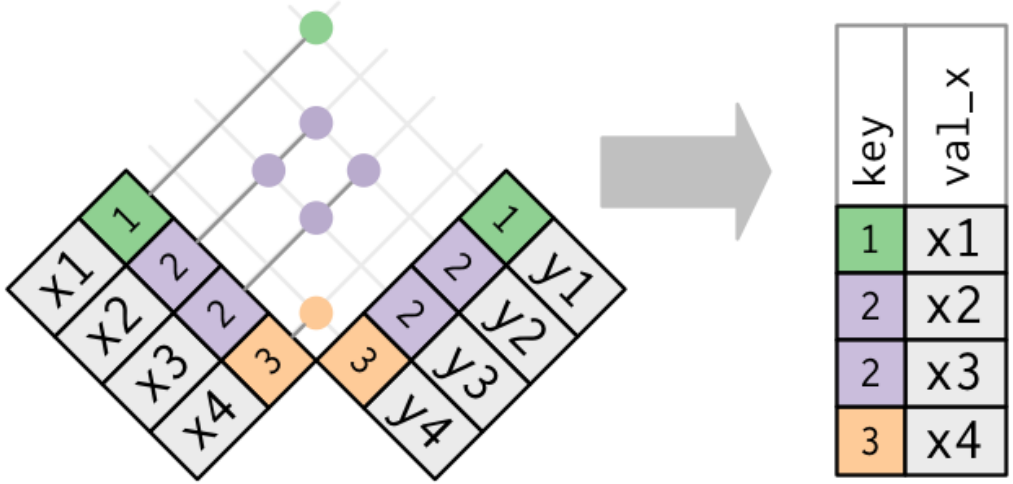
```
flights %>%
  semi_join(top_dest)
#> Joining, by = "dest"
#> # A tibble: 141,145 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>
#>  1  2013     1     1      542            540         2      923
#>  2  2013     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:



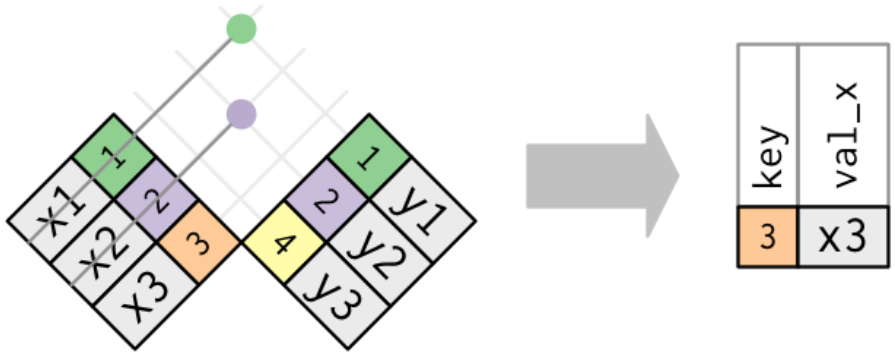
- Many-to-many:



`flights` without a match in `planes`

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
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 NOEGMQ    371
#>  8 N534MQ    364
#>  9 N542MQ    363
#> 10 N531MQ    349
#> # ... with 712 more rows
```



Outline

Warm-up

- Does every year have 365 days?
- Does every day have 24 hours?
- Does every minute have 60 seconds?

Refering to an instant in time

- Three types of date/time data:
  - A **date**.
    - Tibbles print this as `<date>`.
  - A **time** within a day.
    - Tibbles print this as `<time>`.
  - A **date-time** is a date plus a time.
    - Uniquely identifies an instant in time (typically to the nearest second).
    - Tibbles print this as `<dttm>`.
    - Elsewhere in R, `POSIXct`.
- In R:
  - Focus on dates/date-times because no "native" class for times.
  - If you need one, look at the **hms** package.
- **Use the simplest possible data type satisfying your needs!**

Creating date/times

- The **lubridate** package:
  - Makes it easier to work with dates and times in R.
  - Not part of core tidyverse because only needed when working with dates/times.

```
library(lubridate)
today()
#> [1] "2019-11-01"
now()
#> [1] "2019-11-01 14:24:32 EDT"
```

- Three other (usual) ways to create a date/time:
  - From a string.
  - From individual date-time components.
  - From an existing date/time object (i.e., with `as_datetime(today())` or conversely `as_date(now())`).

From a string


```
ymd("2017-01-31")
#> [1] "2017-01-31"
mdy("January 31st, 2017")
#> [1] "2017-01-31"
dmy("31-Jan-2017")
#> [1] "2017-01-31"

ymd_hms("2017-01-31 20:11:59")
#> [1] "2017-01-31 20:11:59 UTC"
mdy_hm("01/31/2017 08:01")
#> [1] "2017-01-31 08:01:00 UTC"
```

- Additionally:

```
ymd(20170131)
#> [1] "2017-01-31"
ymd(20170131, tz = "UTC")
#> [1] "2017-01-31 UTC"
```

From individual components


```
flights %>%
  select(year, month, day, hour, minute, dep_time) %>%
  mutate(departure = make_datetime(year, month, day, hour, minute))
#> # A tibble: 336,776 x 7
#>     year month   day  hour minute dep_time departure
#>    <int> <int> <int> <dbl>  <dbl>    <int> <dttm>
#>  1  2013     1     1     5     15      517 2013-01-01 05:15:00
#>  2  2013     1     1     5     29      533 2013-01-01 05:29:00
#>  3  2013     1     1     5     40      542 2013-01-01 05:40:00
#>  4  2013     1     1     5     45      544 2013-01-01 05:45:00
#>  5  2013     1     1     6      0      554 2013-01-01 06:00:00
#>  6  2013     1     1     5     58      554 2013-01-01 05:58:00
#>  7  2013     1     1     6      0      555 2013-01-01 06:00:00
#>  8  2013     1     1     6      0      557 2013-01-01 06:00:00
#>  9  2013     1     1     6      0      557 2013-01-01 06:00:00
#> 10  2013     1     1     6      0      558 2013-01-01 06:00:00
#> # ... with 336,766 more rows
```

- For dep_time and others such as arr_time:

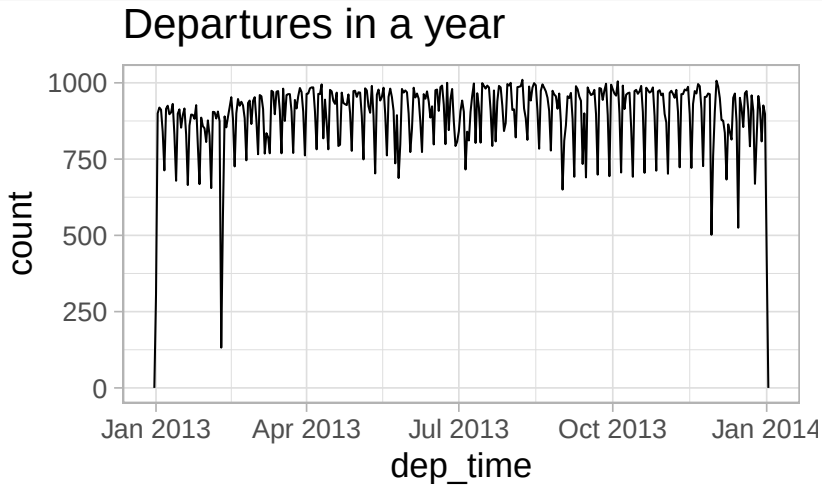
```
make_datetime_100 <- function(year, month, day, time) {
  make_datetime(year, month, day, time %/% 100, time %% 100)
}
```

From individual components II

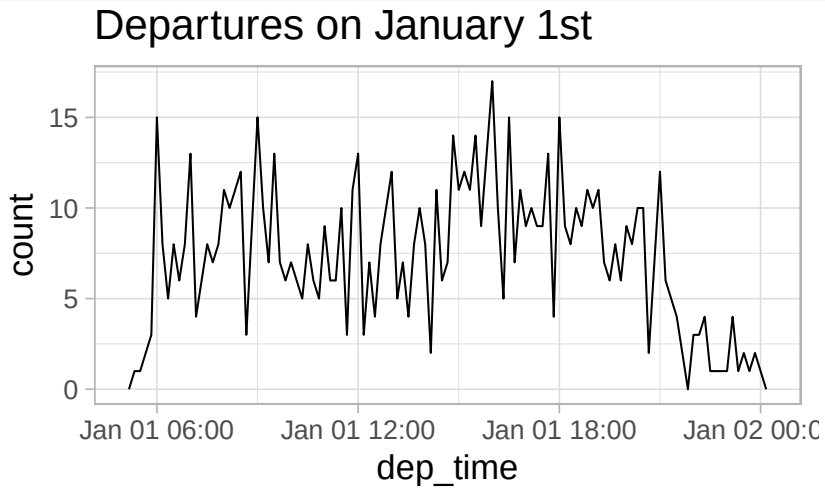
COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```r
flights_dt <- flights %>%
  filter(!is.na(dep_time), !is.na(arr_time)) %>%
  mutate(dep_time = make_datetime_100(year, month, day, dep_time),
         arr_time = make_datetime_100(year, month, day, arr_time)) %>%
  select(origin, dest, ends_with("delay"), ends_with("time"))
```

```r
flights_dt %>%
  ggplot(aes(dep_time)) +
  geom_freqpoly(binwidth = 86400) + # 86400s=1d
  ggtitle("Departures in a year")
#
```



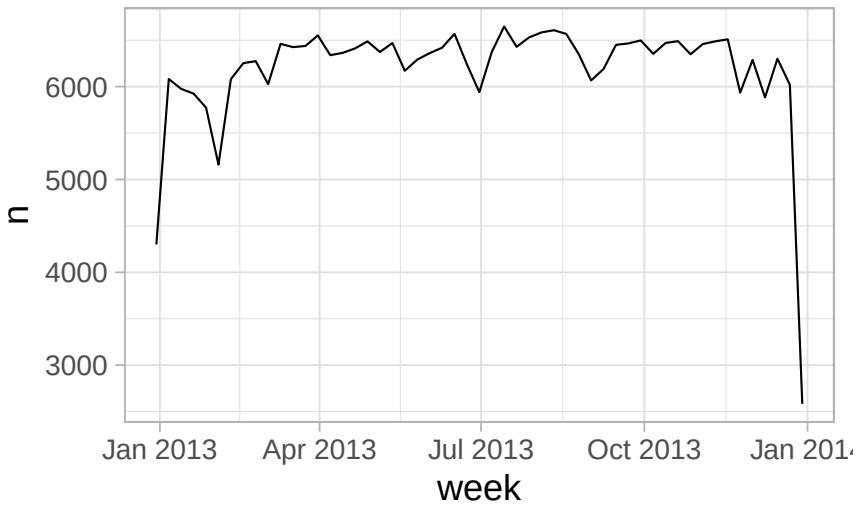

```r
flights_dt %>%
  filter(dep_time < ymd(20130102)) %>%
  ggplot(aes(dep_time)) +
  geom_freqpoly(binwidth = 600) + # 600s=10mn
  ggtitle("Departures on January 1st")
```



Rounding

- Rounding:
  - `floor_date()` rounds down.
  - `round_date()` rounds to.
  - `ceiling_date()` rounds up.
- Takes a vector of dates to adjust and then the name of the unit:

```
flights_dt %>%
  count(week = floor_date(dep_time, "week")) %>%
  ggplot(aes(week, n)) +
  geom_line()
```



Getting/setting the components

- Getting the components:

```
datetime <- ymd_hms("2016-07-08 12:34:56")
c(year(datetime), month(datetime), mday(datetime),
  yday(datetime), wday(datetime))
#> [1] 2016    7    8  190    6
```

- Setting the components:

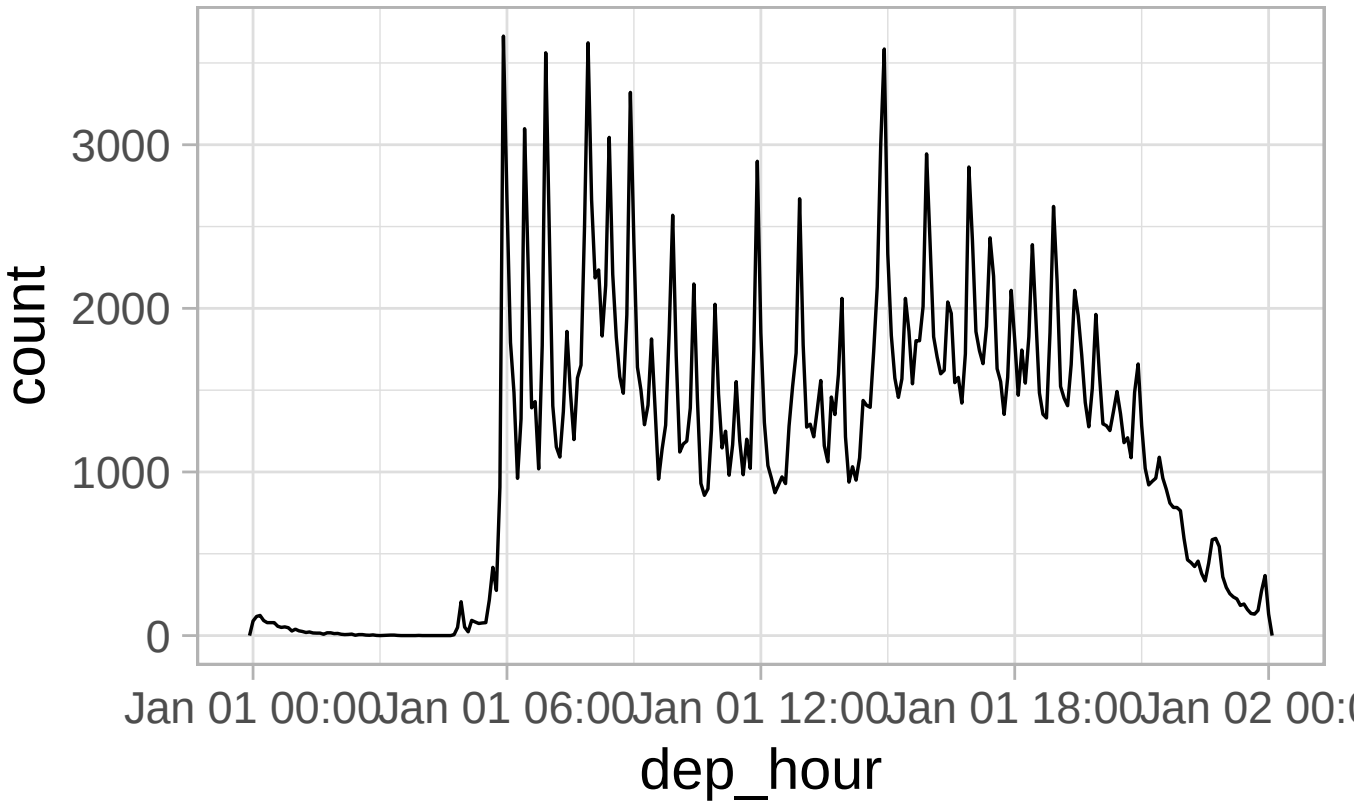
```
year(datetime) <- 2020
datetime
#> [1] "2020-07-08 12:34:56 UTC"
month(datetime) <- 01
datetime
#> [1] "2020-01-08 12:34:56 UTC"
hour(datetime) <- hour(datetime) + 1
datetime
#> [1] "2020-01-08 13:34:56 UTC"
```

- Alternatively, use e.g. `update(datetime, year = 2020)`.

Flights distribution across the day

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
flights_dt %>%
  mutate(dep_hour = update(dep_time, yday = 1)) %>%
  ggplot(aes(dep_hour)) +
    geom_freqpoly(binwidth = 300)
```



Time spans

- Goal: to do arithmetic (i.e., subtraction, addition, and division) with dates/times.
- Three classes that represent time spans:
  - **Durations** (number of seconds).
  - **Periods** (human units like weeks and months).
  - **Intervals** (a starting and ending point).

Durations

- A **duration** always record a time span in seconds.
- Larger units created at the standard rate.
  - E.g., 60s/mn, 60mn/h, 24h/d, 7d/w, 365d/y.

```
dseconds(15)
#> [1] "15s"
dminutes(10)
#> [1] "600s (~10 minutes)"
dhours(c(12, 24))
#> [1] "43200s (~12 hours)" "86400s (~1 days)"
ddays(0:5)
#> [1] "0s"                 "86400s (~1 days)"  "172800s (~2 days)"
#> [4] "259200s (~3 days)" "345600s (~4 days)" "432000s (~5 days)"
dweeks(3)
#> [1] "1814400s (~3 weeks)"
dyears(1)
#> [1] "31536000s (~52.14 weeks)"
```

Durations arithmetics

- Add and multiply durations:

```
2 * dyears(1)
#> [1] "63072000s (~2 years)"
dyears(1) + dweeks(12) + dhours(15)
#> [1] "38847600s (~1.23 years)"
```

- Add and subtract durations to and from dates/datetimes:

```
tomorrow <- today() + ddays(1)
last_year <- today() - dyears(1)
```

- What happens here?

```
one_pm <- ymd_hms("2016-03-12 13:00:00", tz = "America/New_York")
one_pm
#> [1] "2016-03-12 13:00:00 EST"
one_pm + ddays(1)
#> [1] "2016-03-13 14:00:00 EDT"
```

Periods

- Work with "human" times, like days (no fixed length in secs):

```
one_pm
#> [1] "2016-03-12 13:00:00 EST"
one_pm + days(1)
#> [1] "2016-03-13 13:00:00 EDT"
seconds(15)
#> [1] "15S"
minutes(10)
#> [1] "10M 0S"
hours(c(12, 24))
#> [1] "12H 0M 0S" "24H 0M 0S"
days(7)
#> [1] "7d 0H 0M 0S"
months(1:3)
#> [1] "1m 0d 0H 0M 0S" "2m 0d 0H 0M 0S" "3m 0d 0H 0M 0S"
weeks(3)
#> [1] "21d 0H 0M 0S"
years(1)
#> [1] "1y 0m 0d 0H 0M 0S"
```

Periods arithmetics

- Add and multiply periods:

```
10 * (months(6) + days(1))
#> [1] "60m 10d 0H 0M 0S"
days(50) + hours(25) + minutes(2)
#> [1] "50d 25H 2M 0S"
```

- Add periods to dates/datetimes:

```
# A leap year
ymd("2016-01-01") + dyears(1)
#> [1] "2016-12-31"
ymd("2016-01-01") + years(1)
#> [1] "2017-01-01"

# Daylight Savings Time
one_pm + ddays(1)
#> [1] "2016-03-13 14:00:00 EDT"
one_pm + days(1)
#> [1] "2016-03-13 13:00:00 EDT"
```

Intervals

- What should the following code return?

```
years(1) / days(1)
```

- A duration with a starting point:

```
next_year <- today() + years(1)
(today() %--% next_year) / ddays(1)
#> [1] 366
```

Summary

	date				date time				duration				period				interval				number			
date	-								-	+			-	+							-	+		
date time					-				-	+			-	+							-	+		
duration	-	+			-	+			-	+		/									-	+	×	/
period	-	+			-	+							-	+							-	+	×	/
interval												/				/								
number	-	+			-	+			-	+	×		-	+	×		-	+	×		-	+	×	/

- Pick the simplest data structure that solves your problem:
  - If you only care about physical time, use a duration.
  - If you need to add human times, use a period.
  - If you need to figure out how long a span is in human units, use an interval.

Time zones


```
Sys.timezone()
#> [1] "America/New_York"
length(OlsonNames())
#> [1] 607
head(OlsonNames())
#> [1] "Africa/Abidjan"     "Africa/Accra"       "Africa/Addis_Ababa"
#> [4] "Africa/Algiers"     "Africa/Asmara"      "Africa/Asmera"
```

Same instant in different time zones

- Same instant, different place:

```
(x1 <- ymd_hms("2015-06-01 12:00:00", tz = "America/New_York"))
#> [1] "2015-06-01 12:00:00 EDT"
(x2 <- ymd_hms("2015-06-01 18:00:00", tz = "Europe/Copenhagen"))
#> [1] "2015-06-01 18:00:00 CEST"
(x3 <- ymd_hms("2015-06-02 04:00:00", tz = "Pacific/Auckland"))
#> [1] "2015-06-02 04:00:00 NZST"
x1 - x2
#> Time difference of 0 secs
x1 - x3
#> Time difference of 0 secs
```

- Note the behavior of ‘c():

```
x4 <- c(x1, x2, x3)
x4
#> [1] "2015-06-01 12:00:00 EDT" "2015-06-01 12:00:00 EDT"
#> [3] "2015-06-01 12:00:00 EDT"
```

Changing the time zone

- Keep the instant in time:

```
x4a <- with_tz(x4, tzone = "Australia/Lord_Howe")
x4a
#> [1] "2015-06-02 02:30:00 +1030" "2015-06-02 02:30:00 +1030"
#> [3] "2015-06-02 02:30:00 +1030"
x4a - x4
#> Time differences in secs
#> [1] 0 0 0
```

- Change the instant in time:

```
x4b <- force_tz(x4, tzone = "Australia/Lord_Howe")
x4b
#> [1] "2015-06-01 12:00:00 +1030" "2015-06-01 12:00:00 +1030"
#> [3] "2015-06-01 12:00:00 +1030"
x4b - x4
#> Time differences in hours
#> [1] -14.5 -14.5 -14.5
```

Outline

Factors

- Factors are:
  - Used to work with categorical variables (i.e., that have a fixed and known set of possible values).
  - Useful to display character vectors in a non-alphabetical order.
- The **forcats** package:
  - Range of helpers for working with factors.

```
library(forcats)
```

Creating factors

- Imagine that you have a variable that records month:

```
x1 <- c("Dec", "Apr", "Jan", "Mar")
```

- Using a string to record this variable has two problems:
  - Twelve possible months and nothing saving you from typos.
  - It doesn't sort in a useful way.

```
sort(x1)
#> [1] "Apr" "Dec" "Jan" "Mar"
```

Creating factors II

- Start by creating a list of the valid **levels**:

```
month_levels <- c("Jan", "Feb", "Mar", "Apr", "May", "Jun",
                  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")
```

- Then create a factor:

```
y1 <- factor(x1, levels = month_levels)
y1
#> [1] Dec Apr Jan Mar
#> Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
sort(y1)
#> [1] Jan Mar Apr Dec
#> Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
factor(x1) ## without levels
#> [1] Dec Apr Jan Mar
#> Levels: Apr Dec Jan Mar
```

Creating factors III

- Notice:

```
x2 <- c("Dec", "Apr", "Jam", "Mar")
y2 <- factor(x2, levels = month_levels)
y2
#> [1] Dec  Apr  <NA> Mar
#> Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

- Other ordering:

```
factor(x1, levels = unique(x1))
#> [1] Dec Apr Jan Mar
#> Levels: Dec Apr Jan Mar
factor(x1) %>%
  fct_inorder()
#> [1] Dec Apr Jan Mar
#> Levels: Dec Apr Jan Mar
```

forcats::gss_cat

- Sample from the General Social Survey:

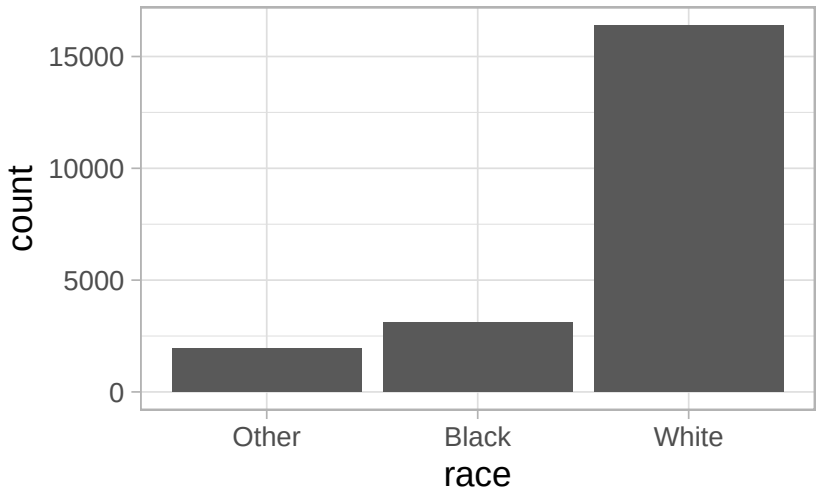
```
gss_cat
#> # A tibble: 21,483 x 9
#>     year marital    age race  rincome  partyid  relig  denom  tvhours
#>    <int> <fct>    <int> <fct> <fct>    <fct>    <fct>  <fct>    <int>
#>  1  2000 Never m~    26 White $8000 t~ Ind,nea~ Prote~ South~      12
#>  2  2000 Divorced    48 White $8000 t~ Not str~ Prote~ Bapti~      NA
#>  3  2000 Widowed     67 White Not app~ Indepen~ Prote~ No de~       2
#>  4  2000 Never m~    39 White Not app~ Ind,nea~ Ortho~ Not a~       4
#>  5  2000 Divorced    25 White Not app~ Not str~ None   Not a~       1
#>  6  2000 Married     25 White $20000 ~ Strong ~ Prote~ South~      NA
#>  7  2000 Never m~    36 White $25000 ~ Not str~ Chris~ Not a~       3
#>  8  2000 Divorced    44 White $7000 t~ Ind,nea~ Prote~ Luthe~      NA
#>  9  2000 Married     44 White $25000 ~ Not str~ Prote~ Other        0
#> 10  2000 Married     47 White $25000 ~ Strong ~ Prote~ South~       3
#> # ... with 21,473 more rows
```

- More info with `?gss_cat`.

See levels of a factor from a tibble

- A barplot:

```
ggplot(gss_cat, aes(race)) +
  geom_bar()
```

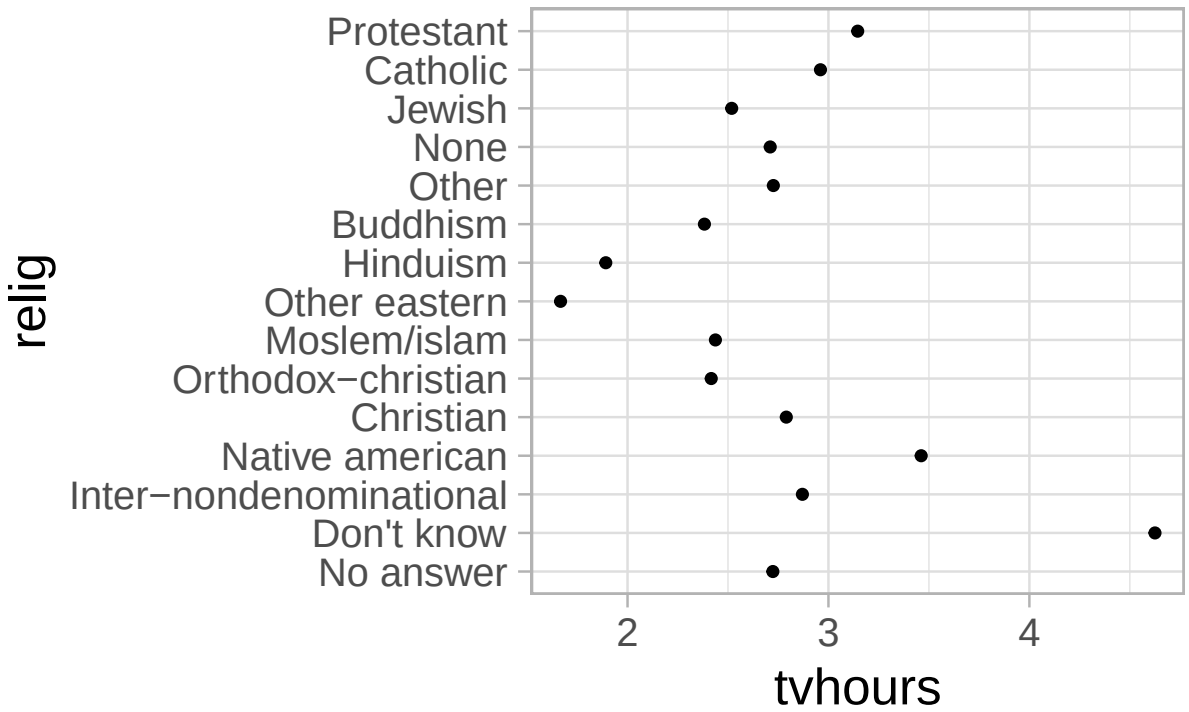


- Or a count:

```
gss_cat %>%
  count(race)
#> # A tibble: 3 x 2
#>   race      n
#>   <fct> <int>
#> 1 Other  1959
#> 2 Black  3129
#> 3 White 16395
```

What's wrong here?

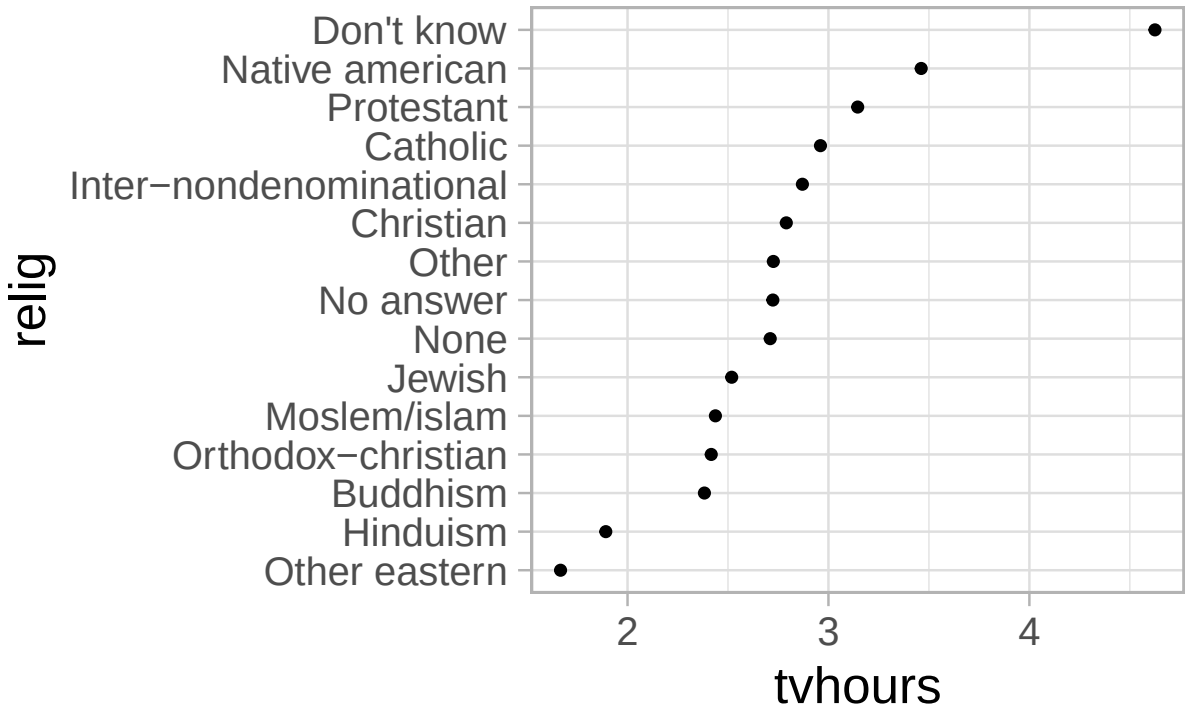

```
relig_summary <- gss_cat %>%
  group_by(relig) %>%
  summarize(age = mean(age, na.rm = TRUE),
            tvhours = mean(tvhours, na.rm = TRUE),
            n = n())

ggplot(relig_summary, aes(tvhours, relig)) +
  geom_point()
```



Modifying factor order

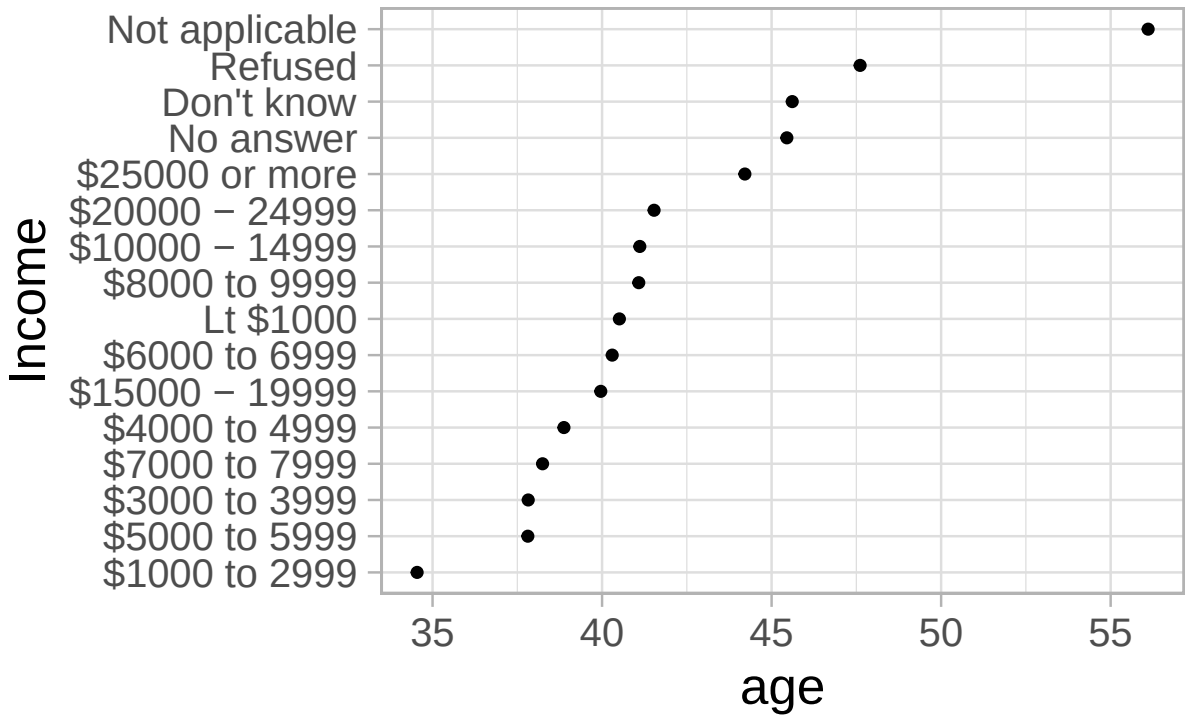
COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
relig_summary %>%
  mutate(relig = fct_reorder(relig, tvhours)) %>%
  ggplot(aes(tvhours, relig)) +
  geom_point()
```



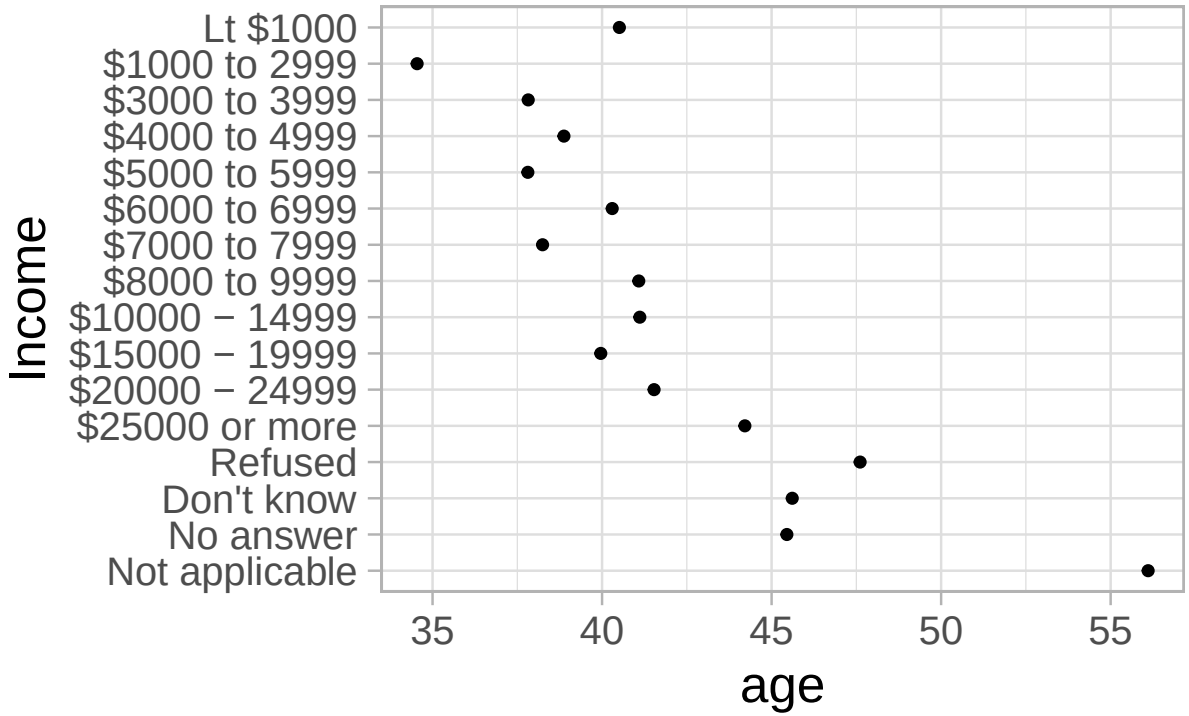
What's wrong here?


```
rincome_summary <- gss_cat %>%
  group_by(rincome) %>%
  summarize(age = mean(age, na.rm = TRUE),
            tvhours = mean(tvhours, na.rm = TRUE),
            n = n())

ggplot(rincome_summary, aes(age, fct_reorder(rincome, age))) +
  geom_point() + ylab("Income")
```



Modify factor order II

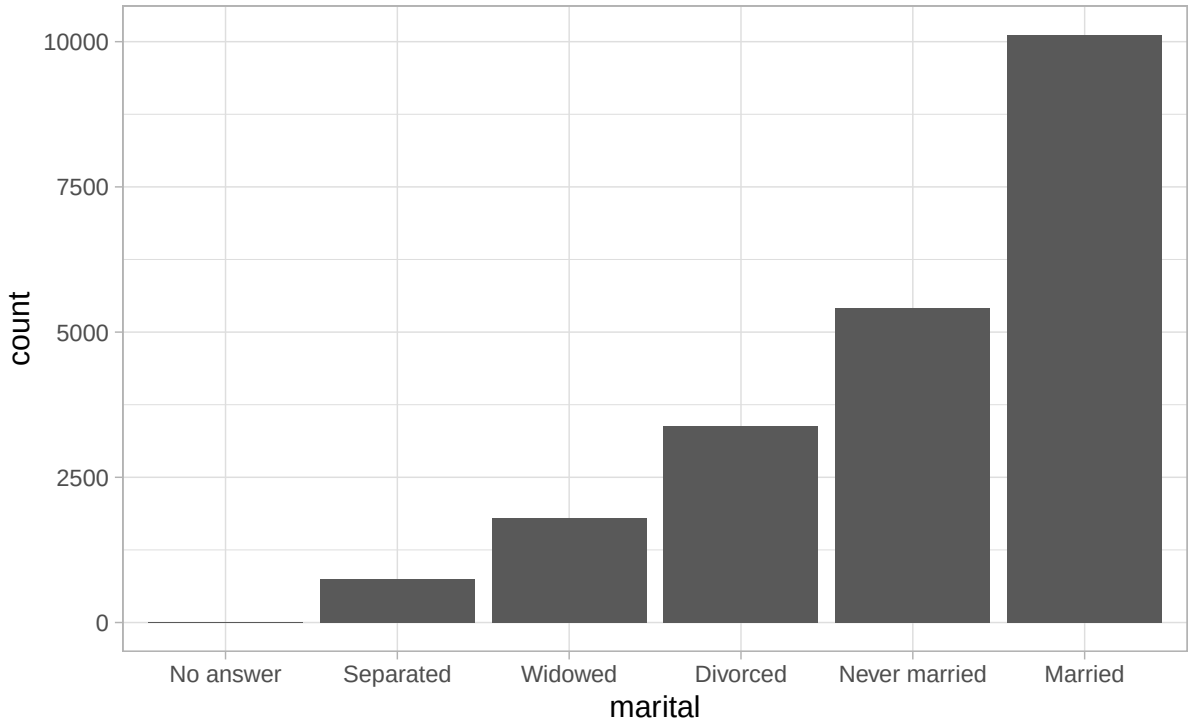

```
ggplot(rincome_summary,
       aes(age, fct_relevel(rincome, "Not applicable"))) +
  geom_point() +
  ylab("Income")
```



- Why do you think the average age for "Not applicable" is so high?

Modify factor order III


```
gss_cat %>%
  mutate(marital = marital %>% fct_infreq() %>% fct_rev()) %>%
  ggplot(aes(marital)) +
  geom_bar()
```



Modifying factor levels

- More powerful than changing the orders of the levels is changing their values:
  - To clarify labels for publication.
  - To collapse levels for high-level displays.

What's wrong here?

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
gss_cat %>%
  count(partyid)
#> # A tibble: 10 x 2
#>    partyid              n
#>    <fct>            <int>
#>  1 No answer          154
#>  2 Don't know           1
#>  3 Other party        393
#>  4 Strong republican 2314
#>  5 Not str republican 3032
#>  6 Ind,near rep       1791
#>  7 Independent        4119
#>  8 Ind,near dem       2499
#>  9 Not str democrat   3690
#> 10 Strong democrat    3490
```

Modifying factor levels II

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
gss_cat %>%
  mutate(partyid = fct_recode(partyid,
    "Republican, strong"    = "Strong republican",
    "Republican, weak"      = "Not str republican",
    "Independent, near rep" = "Ind,near rep",
    "Independent, near dem" = "Ind,near dem",
    "Democrat, weak"        = "Not str democrat",
    "Democrat, strong"      = "Strong democrat")) %>%
  count(partyid)
#> # A tibble: 10 x 2
#>    partyid                  n
#>    <fct>                <int>
#>  1 No answer              154
#>  2 Don't know               1
#>  3 Other party            393
#>  4 Republican, strong    2314
#>  5 Republican, weak      3032
#>  6 Independent, near rep 1791
#>  7 Independent           4119
#>  8 Independent, near dem 2499
#>  9 Democrat, weak        3690
#> 10 Democrat, strong      3490
```

Collapsing factors


```
gss_cat %>%
  mutate(partyid = fct_recode(partyid,
    "Republican, strong"    = "Strong republican",
    "Republican, weak"      = "Not str republican",
    "Independent, near rep" = "Ind,near rep",
    "Independent, near dem" = "Ind,near dem",
    "Democrat, weak"        = "Not str democrat",
    "Democrat, strong"      = "Strong democrat",
    "Other"                 = "No answer",
    "Other"                 = "Don't know",
    "Other"                 = "Other party" )) %>%
  count(partyid)
#> # A tibble: 8 x 2
#>   partyid                n
#>   <fct>                <int>
#> 1 Other                  548
#> 2 Republican, strong    2314
#> 3 Republican, weak      3032
#> 4 Independent, near rep 1791
#> 5 Independent           4119
#> 6 Independent, near dem 2499
#> 7 Democrat, weak        3690
#> 8 Democrat, strong      3490
```

Collapsing factors II

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
gss_cat %>%
  mutate(partyid = fct_collapse(partyid,
    other = c("No answer", "Don't know", "Other party"),
    rep = c("Strong republican", "Not str republican"),
    ind = c("Ind,near rep", "Independent", "Ind,near dem"),
    dem = c("Not str democrat", "Strong democrat")
  )) %>%
  count(partyid)
#> # A tibble: 4 x 2
#>   partyid     n
#>   <fct>   <int>
#> 1 other     548
#> 2 rep      5346
#> 3 ind      8409
#> 4 dem      7180
```

Collapsing factor III


```
gss_cat %>%
  mutate(relig = fct_lump(relig)) %>%
  count(relig)
#> # A tibble: 2 x 2
#>   relig           n
#>   <fct>       <int>
#> 1 Protestant 10846
#> 2 Other      10637

gss_cat %>%
  mutate(relig = fct_lump(relig, n = 3)) %>%
  count(relig, sort = TRUE)
#> # A tibble: 4 x 2
#>   relig           n
#>   <fct>       <int>
#> 1 Protestant 10846
#> 2 Catholic    5124
#> 3 None        3523
#> 4 Other       1990
```

Outline

String basics


```r
library(stringr) # package for string manipulation

# To create strings
string1 <- "This is a string"
string2 <- 'To get a "quote" inside a string, use single quotes'
```

- Backslash as escape character:

```r
double_quote <- "\"" # or '"'
single_quote <- '\'' # or "'"
```

- The printed representation is not the string itself:

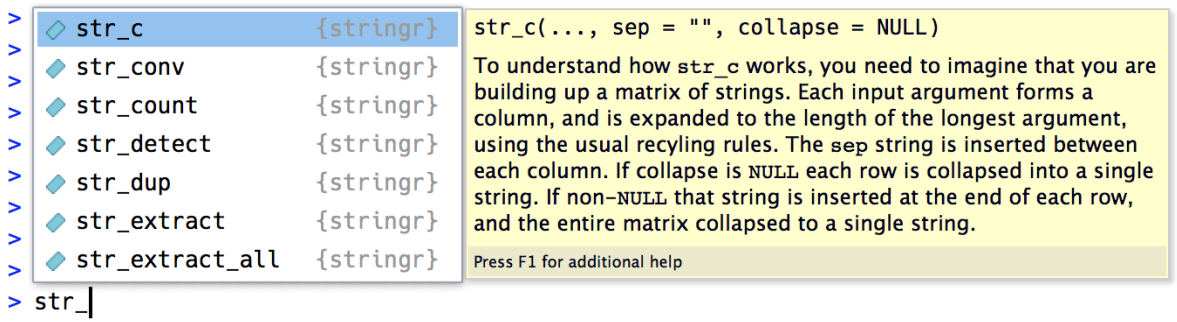
```r
x <- c("\"", "\\")
x
#> [1] "\"" "\\"
writeLines(x)
#> "
#> \
```

More on strings

- Special characters:
  - Use "\n", for newline, or,"\t", for tab.
  - Complete list by requesting help on " (?'"', or ?"'")
- Other usefuls things:

```
(x <- "\u00b5") # Non-English characters
#> [1] "µ"
c("one", "two", "three") # Character vectors
#> [1] "one"   "two"   "three"
str_length(c("a", "R for data science", NA)) # String length
#> [1]  1 18 NA
```

- `stringr` autocomplete:



More on strings II

- Combining strings:

```
str_c("x", "y")
#> [1] "xy"
str_c("x", "y", "z")
#> [1] "xyz"
str_c("x", "y", sep = ", ")
#> [1] "x, y"
```

- Missing values:

```
x <- c("abc", NA)
str_c("|-", x, "-|")
#> [1] "|-abc-|" NA
str_c("|-", str_replace_na(x), "-|")
#> [1] "|-abc-|" "|-NA-|"
```

- Recycling:

```
str_c("prefix-", c("a", "b", "c"), "-suffix")
#> [1] "prefix-a-suffix" "prefix-b-suffix" "prefix-c-suffix"
```

- Collapsing a vector of strings:

```
str_c(c("x", "y", "z"), collapse = ", ")
#> [1] "x, y, z"
```

Subsetting strings


```
x <- c("Apple", "Banana", "Pear")
str_sub(x, 1, 3)
#> [1] "App" "Ban" "Pea"
str_sub(x, -3, -1)
#> [1] "ple" "ana" "ear"
str_sub("a", 1, 5)
#> [1] "a"
str_sub(x, 1, 1) <- str_to_lower(str_sub(x, 1, 1))
x
#> [1] "apple"  "banana" "pear"
```

- See also `str_to_upper()` or `str_to_title()`.

Locales


```
# Turkish has two i's: with and without a dot, and it
# has a different rule for capitalising them:
str_to_upper(c("i", "ı"))
#> [1] "I" "I"
str_to_upper(c("i", "ı"), locale = "tr")
#> [1] "İ" "I"
```

- The locale:
  - An ISO 639 language code, which is a two or three letter abbreviation
  - If blank, R uses the current locale, as provided by your operating system.

Regular expressions

Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now, they have two problems. ——— *Jamie Zawinski*

- A language that allows you to describe patterns in strings.
- Allows you for instance to:
  - Determine which strings match a pattern.
  - Find the positions of matches.
  - Extract the content of matches.
  - Replace matches with new values.
  - Split a string based on a match.
- Read the chapter on strings from the book!

Basic matches

- The simplest patterns match exact strings:

```
x <- c("apple", "banana", "pear")
str_view(x, "an")
```

apple

banana

pear

- Next step is `.`, which matches any character (except a newline):

```
str_view(x, ".a.")
```

apple

banana

pear

- If "." matches any character, how to match the character "."?

Basic matches II

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- If "." matches any character, how to match the character "."?
  - Need to use an "escape" (like string, a backslash \).
  - So to match an ., need the regexp \..
  - But \ is also an escape symbol in strings.
  - So to create the regexp \., use the string "\\.".

```
# To create the regexp, we need \\
dot <- "\\."
# But the expression itself only contains one:
writeLines(dot)
#> \.
# And this tells R to look for an explicit .
str_view(c("abc", "a.c", "bef"), "a\\.c")
```

abc

a.c

bef

Basic matches III

- If \ is an escape character, how do you match a literal \?
  - Need to escape it, i.e. create the regexp \\.
  - To create that regexp with a string, which also needs to escape \, need to write "\\\\"
  - I.e., need four backslashes to match one!

```
x <- "a\\b"
writeLines(x)
#> a\b
str_view(x, "\\\\")
```

Anchors

- By default, regexps match any part of a string.
- Often useful to *anchor* the regexp:
  - ^ to match the start of the string.
  - $ to match the end of the string.

```
x <- c("apple", "banana", "pear")
```

```
str_view(x, "^a")
```

```
apple
banana
pear
```

```
str_view(x, "a$")
```

```
apple
banana
pear
```

- To remember, Evan Misshula's mnemonic: if you begin with power (^), you end up with money ($).

Anchors II

- To force a regexp to only match a complete string, anchor it with both ^ and $:

```
x <- c("apple pie", "apple", "apple cake")
```

```
str_view(x, "apple")
```

apple pie
apple
apple cake

```
str_view(x, "^apple$")
```

apple pie
apple
apple cake

Character classes and alternatives

- Some special patterns match more than one character:
  - Already seen `.`, which matches any character apart from a newline.
  - Two other useful tools:
    - `\d`: matches any digit.
    - `\s`: matches any whitespace (e.g. space, tab, newline).
  - To create a regexp containing `\d` or `\s`:
    - Need to escape the `\` for the string.
    - So type `"\\d"` or `"\\s"`.
- The other two tools are:
  - **Character classes**
    - `[abc]`: matches a, b, or c.
    - `[^abc]`: matches anything except a, b, or c.
  - **Alternatives**
    - `abc|d..f`: matches either `"abc"`, or `"deaf"`.

Character classes

- Can be used as an alternative to backslash escapes.

```
str_view(c("abc",
           "a.c",
           "a*c",
           "a c"),
         "a[.]c")
```

abc

a.c

a*c

a c

```
str_view(c("abc",
           "a.c",
           "a*c",
           "a c"),
         ".[*]c")
```

abc

a.c

a*c

a c

```
str_view(c("abc",
           "a.c",
           "a*c",
           "a c"),
         "a[ ]")
```

abc

a.c

a*c

a c

- Used to pick between one or more alternative patterns.
- Works for most regex metacharacters: `$ . | ? * + ( ) [ {`.
- But some have special meaning even inside a character class.
  - Must be handled with backslash escapes: `] \ ^` and `-`.

Alternatives

- Note that the precedence for `|` is low:
  - `abc|xyz`: matches `abc` or `xyz`, not `abcyz` or `abxyz`.
- Same as mathematical expressions: if it gets confusing, use parentheses.

```
str_view(c("grey", "gray"), "gr(e|a)y")
```

grey

gray

Repetition

- To control how many times a pattern matches:
  - ?: 0 or 1.
  - +: 1 or more.
  - *: 0 or more.

```
# 1888 is the longest year in Roman numerals
x <- "MDCCCLXXXVIII"
```

```
str_view(x, "CC?")
```

MDCCCLXXXVIII

```
str_view(x, "CC+")
```

MDCCCLXXXVIII

```
str_view(x, 'C[LX]+')
```

MDCCCLXXXVIII

- The precedence of these operators is high:
  - `colou?r`: matches either US or British spellings.
  - Most uses will need parentheses, like `bana(na)+`.

Repetition

- To specify the number of matches precisely:
  - `{n}`: exactly n.
  - `{n,}`: n or more.
  - `{,m}`: at most m.
  - `{n,m}`: between n and m.

```
str_view(x, "C{2}")
```

MDCCCLXXXVIII

```
str_view(x, "C{2,}")
```

MDCCCLXXXVIII

```
str_view(x, "C{2,3}")
```

MDCCCLXXXVIII

Grouping and backreferences

- Earlier: parentheses as a way to disambiguate complex expressions.
- But parentheses also create a *numbered* capturing group.
- A capturing group stores *the part of the string* matched by the part of the regexp inside the parentheses.
- Refer to the same text as previously matched by a capturing group with *backreferences*, like \1, \2 etc.

```
str_view(fruit, "(..)\\1", match = TRUE)
```

```
banana
coconut
cucumber
jujube
papaya
salal berry
```

- Cool applications in chapter 14.4!