COMP824 2023 Week 9
Relational Data

Department of Mathematical Sciences
Auckland University of Technology

Overview

Relational data

Keys

Joins

Visualising Geographic Data

Reading

Chapter 13 Wickham and Grolemund (2020), R for Data Science https://r4ds.had.co.nz/

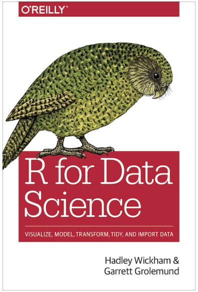
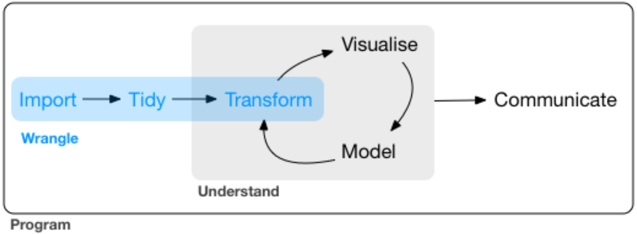


Figure 1: http://r4ds.had.co.nz/

The Process of Analytics



Learning objectives

- Recognise relational data
- Understand the main types of mutating and filtering joins
- Join datasets using appropriate `tidyverse` join functions

Relational data

Multiple tables of related data = **relational data**

Example

- Each flight has an airline.
- Each airline has multiple flights
- The tibbles `airlines` and `flights` are related.

Flights data

```r
nycflights13::flights
```

```
# A tibble: 336,776 x 19
   year month   day dep_time sched~1 dep_d~2 arr_t~3 sched~4
  <int> <int> <int>    <int>   <int>   <dbl>   <int>   <int>
1  2013     1     1      517     515       2     830     819
2  2013     1     1      533     529       4     850     830
3  2013     1     1      542     540       2     923     850
# ... with 336,773 more rows, 11 more variables:
#   arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>, and abbreviated
#   variable names 1: sched_dep_time, 2: dep_delay,
#   3: arr_time, 4: sched_arr_time
```

Airline data

```
nycflights13::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.
# ... with 13 more rows
```

Other related datasets

```r
library(nycflights13)
airlines
flights
planes
airports
weather
```

Digrammatic Representation of nycflights13 datasets

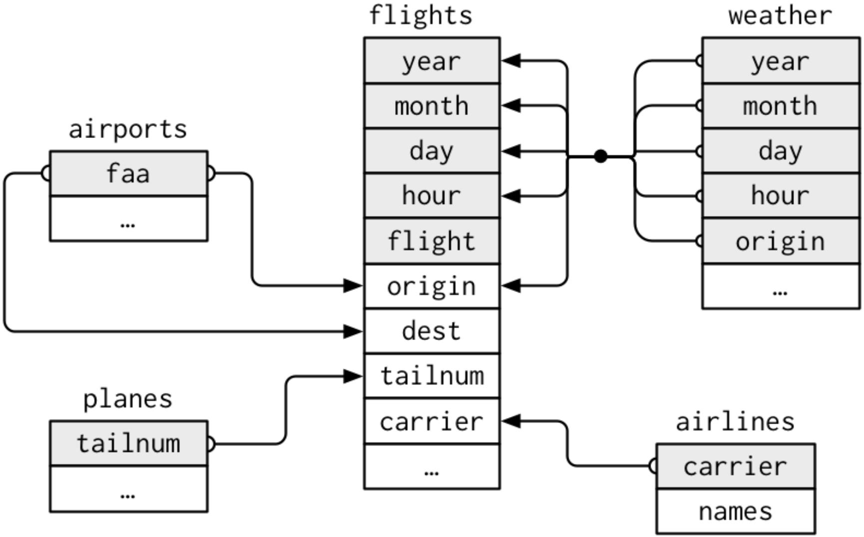


Figure 3: https://r4ds.had.co.nz/relational-data.html

Keys

Relational data

Keys

Joins

Visualising Geographic Data

Keys

Keys: variables used for connecting pairs of tables

- Primary key: variable/s which uniquely identify observations in their own table
  - weather: year, month, day, hour, origin
  - airports: faa
  - planes: tailnum
- Foreign key: variable/s which unique identify observations in another table
  - `flights$tailnum` is foreign key because it is a primary key in the table `planes`

Keys: Good practice

Good practice

- Identify primary key
- Check they uniquely identify observations
- If no primary key exists add a **surrogate key**

Example: Check key uniquely identifies observations 1

```r
planes %>%
  count(tailnum) %>%
  filter(n > 1)  # unique
```

```
# A tibble: 0 x 2
# ... with 2 variables: tailnum <chr>, n <int>
```

Example: Check key uniquely identifies observations 2

```r
weather %>%
  count(year, month, day, hour, origin) %>%
  filter(n > 1) # not unique
```

```
# A tibble: 3 x 6
   year month   day  hour origin     n
  <int> <int> <int> <int> <chr>  <int>
1  2013    11     3     1 EWR        2
2  2013    11     3     1 JFK        2
3  2013    11     3     1 LGA        2
```

Example: Adding a surrogate key 1

```r
(tb <- tibble(x=c("A","B","B"), y=c(4,6,6)) )
```

```
# A tibble: 3 x 2
  x         y
  <chr> <dbl>
1 A         4
2 B         6
3 B         6
```

```r
tb %>% count(x, y)%>% filter(n > 1)
```

```
# A tibble: 1 x 3
  x         y     n
  <chr> <dbl> <int>
1 B         6     2
```

Example: Adding a surrogate key 2

```r
tb %>% mutate(surrogate_key = row_number())
```

```
# A tibble: 3 x 3
  x         y surrogate_key
  <chr> <dbl>         <int>
1 A         4             1
2 B         6             2
3 B         6             3
```

Joins

Joins

- **Mutating joins**: add new variables from a data frame to another
- **Filtering joins**: filters a data frame based on whether they match another data frame

Relational data is usually stored in a **relational database management system (RDBMS)**.

Mutating Joins: Example

```
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
# ... with 336,773 more rows
```

Suppose we want to add the airline name to the dataset.

Example

```r
flights2 %>%
  # remove columns for easier printing
  select(-origin, -dest) %>%
  # left join by carrier code
  left_join(airlines, by = "carrier")
```

Types of mutating joins

- Inner joins – keeps only observations in x **and** y
- Outer joins
  - Left join – keeps all observations in x
  - Right join – keeps all observations in y
  - Full join – keeps all observations in x and y

Types of mutating joins

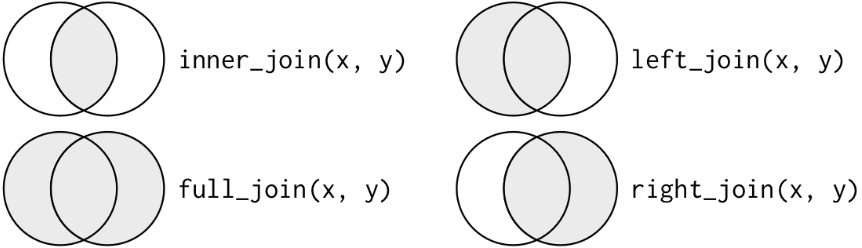


Figure 4: https://r4ds.had.co.nz/relational-data.html

Example

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

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

Inner Joins

- New data has observations in **both** data sets.
- Unmatched observations are not included

```
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: Left Join

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

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

Outer Joins: Right join

```r
x %>%
  right_join(y, by="key")
```

```
# A tibble: 3 x 3
    key val_x val_y
  <dbl> <chr> <chr>
1     1 x1    y1
2     2 x2    y2
3     4 <NA>  y3
```

Outer Joins: Full join

```r
x %>%  full_join(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     3 x3    <NA>
4     4 <NA>  y3
```

Ways of defining the key column

- Natural join: joins by all columns that appear in both tables `x %>% left_join(y, by = NULL)`

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

```
# A tibble: 336,776 x 18
   year month   day  hour origin dest  tailnum carrier  temp
  <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <dbl>
1  2013     1     1     5 EWR    IAH   N14228  UA       39.0
2  2013     1     1     5 LGA    IAH   N24211  UA       39.9
3  2013     1     1     5 JFK    MIA   N619AA  AA       39.0
# ... with 336,773 more rows, and 9 more variables:
#   dewp <dbl>, humid <dbl>, wind_dir <dbl>,
#   wind_speed <dbl>, wind_gust <dbl>, precip <dbl>,
#   pressure <dbl>, visib <dbl>, time_hour <dttm>
```

Ways of defining the key column

- Specify key column using: `by`

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

```
# A tibble: 336,776 x 16
  year.x 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
# ... with 336,773 more rows, and 8 more variables:
#   year.y <int>, type <chr>, manufacturer <chr>,
#   model <chr>, engines <int>, seats <int>, speed <int>,
#   engine <chr>
```

Ways of defining the key column

- Same variable with different names in each table
  - `x %>% left_join(y, by=c("a" = "b"))`
  - Matches `x$a` with `y$b`

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

```
# A tibble: 336,776 x 15
   year month   day  hour origin dest  tailnum carrier name
  <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <chr>
1  2013     1     1     5 EWR    IAH   N14228  UA      Geor~
2  2013     1     1     5 LGA    IAH   N24211  UA      Geor~
3  2013     1     1     5 JFK    MIA   N619AA  AA      Miam~
# ... with 336,773 more rows, and 6 more variables:
#   lat <dbl>, lon <dbl>, alt <dbl>, tz <dbl>, dst <chr>,
#   tzone <chr>
```

Filtering Joins

- `semi_join(x, y)` keep all in `x` that have match in `y`
- `anti_join(x, y)` drop all in `x` that have match in `y`

Similar to `filter`, but `anti-join` and `semi-join` scale better to use with more variables.

Filtering Joins: Example

```r
(exams <- tribble(~studID, ~grade,
                   1, "A",
                   2, "B",
                   3, "C+",
                   5, "D" ))
```

```
# A tibble: 4 x 2
  studID grade
   <dbl> <chr>
1      1 A
2      2 B
3      3 C+
4      5 D
```

Filtering Joins: Example

```
(classlist <- tribble( ~studentID, ~name,
                        1, "Charlotte",
                        2, "Zoe",
                        3, "Caitlin",
                        4, "Abel" ))
```

```
# A tibble: 4 x 2
  studentID name
      <dbl> <chr>
1         1 Charlotte
2         2 Zoe
3         3 Caitlin
4         4 Abel
```

Filtering Joins: Example

Questions of interest

1. Which students in the class sat the exam?
2. Which students didn't sit the exam?
3. Did any students not in the class sit the exam, what was their grade?

Filtering Joins: Semi-joins and Anti-joins

1. Which students in the class sat the exam?

```
classlist %>%
  semi_join(exams, by=c("studentID"="studID"))
```

```
# A tibble: 3 x 2
  studentID name
      <dbl> <chr>
1         1 Charlotte
2         2 Zoe
3         3 Caitlin
```

Filtering Joins: Semi-joins and Anti-joins

2. Which students didn't sit the exam?

```
classlist %>%
  anti_join(exams, by=c("studentID"="studID"))
```

```
# A tibble: 1 x 2
  studentID name
      <dbl> <chr>
1         4 Abel
```

Filtering Joins: Semi-joins and Anti-joins

3. Did any students not in the class sit the exam, what was their grade?

```
exams %>%
  anti_join(classlist, by=c("studID"="studentID"))
```

```
# A tibble: 1 x 2
  studID grade
   <dbl> <chr>
1      5 D
```

Example: Flights Data – Top 10 destinations

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

```
# A tibble: 10 x 2
   dest      n
   <chr> <int>
 1 ORD   17283
 2 ATL   17215
 3 LAX   16174
 4 BOS   15508
 5 MCO   14082
 6 CLT   14064
 7 SFO   13331
 8 FLL   12055
 9 MIA   11728
10 DCA    9705
```

Filtering Joins: Semi-joins

1. **Find all flights to the top destinations.**

```
flights %>%    semi_join(top_dest)
```

```
# A tibble: 141,145 x 19
   year month   day dep_time sched~1 dep_d~2 arr_t~3 sched~4
  <int> <int> <int>    <int>   <int>   <dbl>   <int>   <int>
1  2013     1     1      542     540       2     923     850
2  2013     1     1      554     600      -6     812     837
3  2013     1     1      554     558      -4     740     728
# ... with 141,142 more rows, 11 more variables:
#   arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>, and abbreviated
#   variable names 1: sched_dep_time, 2: dep_delay,
#   3: arr_time, 4: sched_arr_time
```

Filtering Joins: Semi-joins

1. **Find all flights to the top destinations.**

```
flights %>%     semi_join(top_dest)
```

Equivalent to:

```
flights %>%     filter(dest %in% top_dest$dest)
```

Filtering Joins: Anti-joins

2. **Find flights whose plane isn't in `planes`**

```
flights %>%
  anti_join(planes, by = "tailnum") %>%
  count(tailnum, sort = TRUE)
```

```
# A tibble: 722 x 2
  tailnum     n
  <chr>   <int>
1 <NA>     2512
2 N725MQ    575
3 N722MQ    513
# ... with 719 more rows
```

Filtering Joins: Anti-joins

2. **Find flights whose plane isn't in `planes`**

```
flights %>%
  anti_join(planes, by = "tailnum") %>%
  count(tailnum, sort = TRUE)
```

Equivalent to:

```
flights %>%
  filter(! tailnum %in% planes$tailnum) %>%
  count(tailnum, sort = TRUE)
```

Further topics in relational data

- Duplicate keys
- Join problems
- Set operations

If you are going to be working with relational data, you should read up about these topics. See Chapter 13 Wickham and Grolemund (2020).

Visualising Geographic Data

- Relational data
- Keys
- Joins
- Visualising Geographic Data

Join `flights` and `airports`

- Add longitude and latitude of airports to flights data

```r
flights_loc <- flights %>%
  select(carrier, flight, tailnum, origin, dest) %>%
  inner_join(select(airports, faa, name, lat, lon),
             by = c("origin"="faa")) %>%
  inner_join(select(airports, faa, name, lat, lon),
             by = c("dest"="faa"),
             suffix = c("_origin", "_dest"))
```

Application: Join `flights` and `airports`

```
flights_loc
```

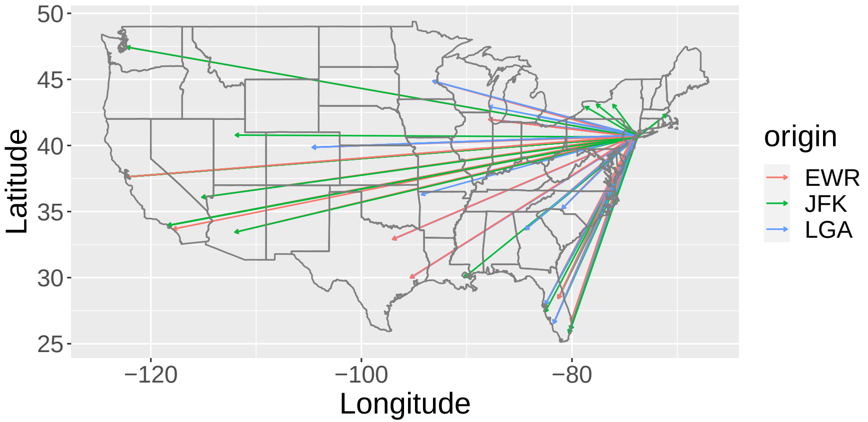
```
# A tibble: 329,174 x 11
  carrier flight tailnum origin dest  name_origin    lat_o~1
  <chr>    <int> <chr>   <chr>  <chr> <chr>            <dbl>
1 UA        1545 N14228  EWR    IAH   Newark Libert~    40.7
2 UA        1714 N24211  LGA    IAH   La Guardia        40.8
3 AA        1141 N619AA  JFK    MIA   John F Kenned~    40.6
# ... with 329,171 more rows, 4 more variables:
#   lon_origin <dbl>, name_dest <chr>, lat_dest <dbl>,
#   lon_dest <dbl>, and abbreviated variable name
#   1: lat_origin
```

Visualise Flight Paths

```
flight_paths_plot <- flights_loc %>%
  slice_head(n= 100) %>%
  ggplot() +
  geom_segment(mapping = aes(
    x = lon_origin, xend = lon_dest,
    y = lat_origin, yend = lat_dest,
    col = origin),
    arrow = arrow(length = unit(0.1, "cm"))) +
    borders(database = "state") +
    #borders(database = "world") +
  coord_quickmap() +
  labs(y = "Latitude", x = "Longitude")
```

Visualise Flight Paths (n = 100 flights)

```
flight_paths_plot
```



Most common destinations

```
(dest_freq <- flights %>%
  count(dest) %>%
  inner_join(airports, by=c("dest"="faa")) %>%
   arrange(-n) )
```

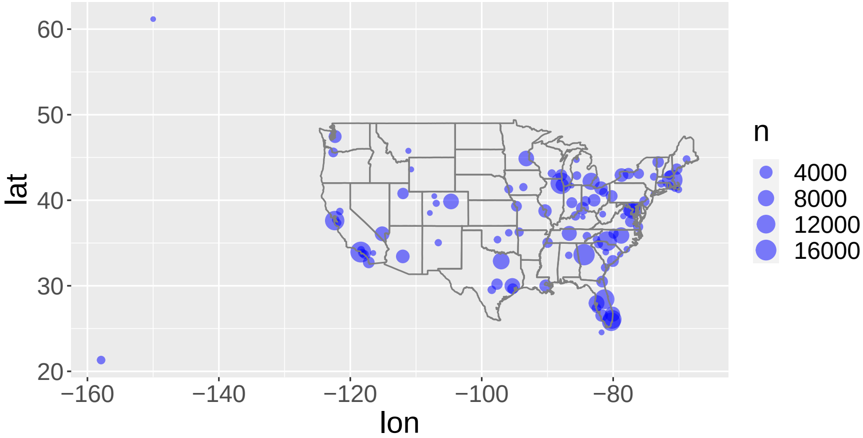
```
# A tibble: 101 x 9
  dest      n name          lat    lon   alt    tz dst   tzone
  <chr> <int> <chr>       <dbl>  <dbl> <dbl> <dbl> <chr> <chr>
1 ORD   17283 Chicago ~    42.0  -87.9   668    -6 A     Amer~
2 ATL   17215 Hartsfie~    33.6  -84.4  1026    -5 A     Amer~
3 LAX   16174 Los Ange~    33.9 -118.    126    -8 A     Amer~
# ... with 98 more rows
```

Plot of most common destinations from NYC airports

```
common_dest_plot <- ggplot() +
  geom_point(data = dest_freq,
             aes(x = lon , y = lat, size = n),
             alpha = 0.5, col = "blue") +
  borders(database = "state") +
  #borders(database = "world") +
  coord_fixed(1.3) +
  guides(fill=FALSE, scale = "none")
```

Plot of most common destinations from NYC airports

```
common_dest_plot
```

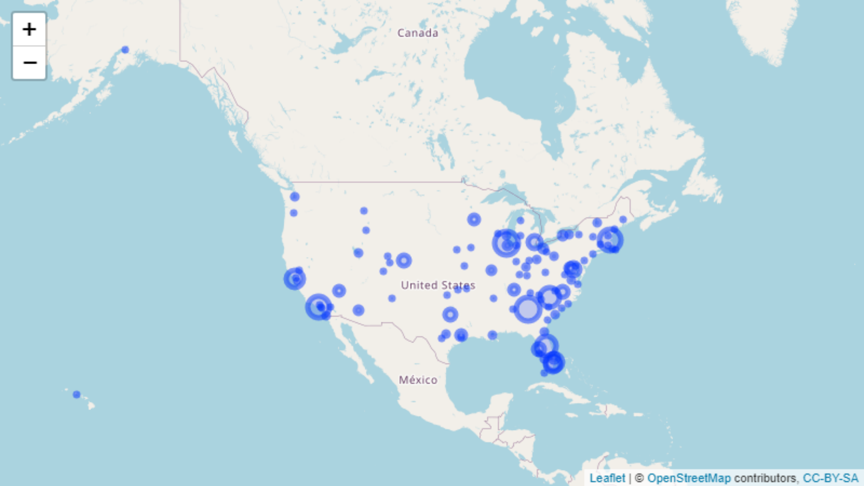


Leaflet plot

```
library(leaflet)
m <- leaflet(dest_freq) %>%
  addTiles() %>%
  #addProviderTiles(providers$Esri.WorldImagery) %>%
  addCircleMarkers(lng = ~lon,                   lat = ~lat,
                   popup = ~as.character(name),
                   label = ~as.character(name),
                   radius = ~n/1600) %>%
  setView(lng = -100, lat = 42, zoom = 3)

# save html widget as image
htmlwidgets::saveWidget(m, "temp.html", selfcontained=TRUE)
webshot2::webshot("temp.html",
                  file="Rfigs/leaflet_map.png",
                  cliprect="viewport",
                  vwidth = 800, vheight = 450)
```

Leaflet plot



Summary

Relational data

Keys

Joins

Visualising Geographic Data

Learning objectives

- Recognise relational data
- Understand the main types of mutating and filtering joins
- Join datasets using appropriate `tidyverse` join functions

References

Wickham, Hadley, and Garrett Grolemund. 2020. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*.