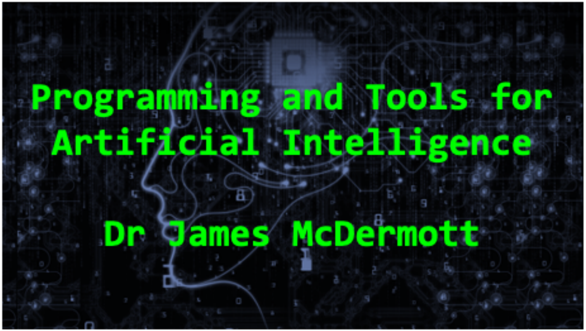
dplyr joins

James McDermott

University of Galway


Programming and Tools for
Artificial Intelligence
Dr James McDermott

`dplyr` joins

Relational databases

The main ideas of relational databases (SQL) are probably familiar to all:

- A database consists of tables
- A table consists of a set of columns
- A column has a type, and maybe some constraints (e.g. positive integer)
- Some column(s) may be designated as a key for the table

Joins

- As we know, in Relational Databases, it is good practice to use *normalisation*: splitting a table up into multiple tables, to avoid duplication of information and the possibility of *update anomalies*. 3NF is the result of normalisation.
- Doing ML/stats/analytics may require *de-normalisation* – re-joining – eventually to export to our ML/stats/analytics system.

Before normalisation

Movie rental DB				
Date	**Movie**	**Genre**	**Customer**	**Address**
01-Jan	Amelie	Romance	Bob	11, Haight St
02-Jan	The Matrix	Sci-fi	Frida	Oxford Circus
02-Jan	Amelie	Romance	Carrie	99, Fifth Ave
05-Jan	Skyfall	Adventure	Bob	11, Haight St
05-Jan	Avengers	Sci-fi	Frida	Oxford Circus

After normalisation: 3rd Normal Form (3NF)

Rentals table

Date	**Movie ID**	**Customer ID**
01-Jan	102	1
02-Jan	101	2
02-Jan	102	3
05-Jan	103	1
05-Jan	104	2

Customer table

Customer ID	**Name**	**Address**
1	Bob	11, Haight St
2	Frida	Oxford Circus
3	Carrie	99, Fifth Ave

Movie table

Movie ID	**Name**	**Genre**
101	The Matrix	Sci-fi
102	Amelie	Romance
103	Skyfall	Adventure
104	Avengers	Sci-fi

Key columns

After normalisation, the link between data is via key columns – in this case, the Customer ID and Movie ID columns. It is possible to put the original table back together using a **join**. We say that we join **on** the key column.

SQL

In SQL, a JOIN might be something like this. This is an *implicit join*:

```
SELECT * FROM RENTALS, CUSTOMER
WHERE RENTALS.CustomerID = CUSTOMER.CustomerID;
```

This is an equivalent *explicit join*:

```
SELECT * FROM RENTALS JOIN CUSTOMER
ON RENTALS.CustomerID = CUSTOMER.CustomerID;
```

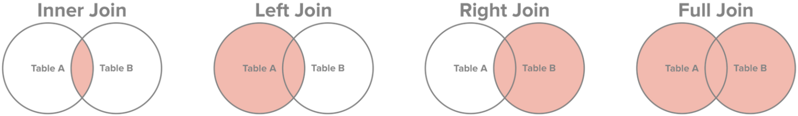
(This is not examinable.)

What is a join, really?

- Think of join as an *operator* whose left and right operands are tables, and whose result is a table formed as the union of their columns
- The *Cross join* is a good place to start. Conceptually, a cross join is a *Cartesian product of rows*. For every row in T1, we put it side by side with every row in T2. Think of that as a new joined table. Now we can select columns from it and filter rows using ON. In particular, we'll probably filter for rows where a key column in one table matches a key column in the other, discarding the large majority of this cross product.
- Other joins just restrict the "every row in T1" and "every row in T2" parts depending on which matches actually exist.

Different types of joins

There are a few types of joins. To distinguish them, many textbooks and cheatsheets proceed to Venn diagrams,
e.g. http://www.sql-join.com/sql-join-types (below). These are helpful as mnemonics but the language of Venn diagrams is not sufficient to define the different joins.



Different types of joins (Data Wrangling Cheatsheet)

a

x1	x2
A	1
B	2
C	3

+

b

x1	x3
A	T
B	F
D	T

=

Mutating Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NA

dplyr::left_join(a, b, by = "x1")

Join matching rows from b to a.

x1	x3	x2
A	T	1
B	F	2
D	T	NA

dplyr::right_join(a, b, by = "x1")

Join matching rows from a to b.

x1	x2	x3
A	1	T
B	2	F

dplyr::inner_join(a, b, by = "x1")

Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NA
D	NA	T

dplyr::full_join(a, b, by = "x1")

Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

dplyr::semi_join(a, b, by = "x1")

All rows in a that have a match in b.

x1	x2
C	3

dplyr::anti_join(a, b, by = "x1")

All rows in a that do not have a match in b.

Further reading

- Most people working in industry in the fields of AI, ML, Data Science, Statistics, etc., use relational databases and SQL a lot.
- We don't teach it, because it is usually seen as a topic for undergrad level. This MOOC is recommended as an optional catch-up or refresher:
  - Stanford Databases https://lagunita.stanford.edu/courses/DB/2014/SelfPaced/about
  - (The following topics in the MOOC are recommended for a "short version": Introduction, JSON, Relational Algebra (Section 1), SQL, Relational Design Theory (Section 1), Unified Modelling Language, Online Analytical Processing)

Exercises

1. Read the three data files `rentals.csv`, `movies.csv`, `customers.csv`, all in the `data/` directory, as tibbles.
2. Optional: get R to read the Date column correctly. Hint: https://readr.tidyverse.org/reference/parse_datetime.html
3. Using a `dplyr` join command, create a table showing the customer name and address for every rental.
4. Piping the result into another join command, recreate the full original table as shown under "Before Normalisation" above.
5. Notice the columns `Name.x` and `Name.y` which appear because there is a `Name` column in each of the Movies and Customers tables. Rename them.
6. Calculate the number of movies Frida watched of the Sci-fi genre.

Solutions

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------- tidyverse
## v ggplot2 3.2.1     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
## -- Conflicts ------------------------------ tidyverse_confli
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Exercises 1 and 2:

```r
rentals <- read_csv("data/rentals.csv",
                    col_types=cols(Date=col_date(
                      format="%d-%b-%Y")))
movies <- read_csv("data/movies.csv")
```

```
## Parsed with column specification:
## cols(
##   MovieID = col_double(),
##   Name = col_character(),
##   Genre = col_character()
## )
```

```r
customers <- read_csv("data/customers.csv")
```

```
## Parsed with column specification:
## cols(
##   CustomerID = col_double(),
##   Name = col_character(),
##   Address = col_character()
```

Customer name and address for each rental

```r
inner_join(rentals, customers, by="CustomerID")
```

```
## # A tibble: 5 x 5
##   Date       MovieID CustomerID Name    Address
##   <date>       <dbl>      <dbl> <chr>   <chr>
## 1 2018-01-01     102          1 Bob     11, Haight St
## 2 2018-01-02     101          2 Frida   Oxford Circus
## 3 2018-01-02     102          3 Carrie  99, Fifth Ave
## 4 2018-01-05     103          1 Bob     11, Haight St
## 5 2018-01-05     104          2 Frida   Oxford Circus
```

Recreate original table

```
inner_join(rentals, customers, by="CustomerID") %>%
  inner_join(movies, by="MovieID")
```

```
## # A tibble: 5 x 7
##   Date       MovieID CustomerID Name.x Address        Name.y
##   <date>       <dbl>      <dbl> <chr>  <chr>          <chr>
## 1 2018-01-01     102          1 Bob    11, Haight St  Amelie
## 2 2018-01-02     101          2 Frida  Oxford Circus  The Ma
## 3 2018-01-02     102          3 Carrie 99, Fifth Ave  Amelie
## 4 2018-01-05     103          1 Bob    11, Haight St  Skyfal
## 5 2018-01-05     104          2 Frida  Oxford Circus  Avenge
```

Rename columns

```
t = inner_join(rentals, customers, by="CustomerID") %>%
  inner_join(movies, by="MovieID") %>%
  rename(CustomerName=Name.x, MovieTitle=Name.y)
t
```

```
## # A tibble: 5 x 7
##   Date       MovieID CustomerID CustomerName Address      Mo
##   <date>       <dbl>      <dbl> <chr>        <chr>        <c
## 1 2018-01-01     102          1 Bob          11, Haight~ An
## 2 2018-01-02     101          2 Frida        Oxford Cir~ Th
## 3 2018-01-02     102          3 Carrie       99, Fifth ~ An
## 4 2018-01-05     103          1 Bob          11, Haight~ Sk
## 5 2018-01-05     104          2 Frida        Oxford Cir~ Av
```

Filter and count

```
t %>% filter(CustomerName=="Frida", Genre=="Sci-fi") %>%
  count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1     2
```

Filter and count

The following is a solution to the problem, but it requires the programmer to do all the work in their head. That's not scalable or flexible and it's error-prone, so don't do this.

```
# Frida is CustomerID 2
# Movies 101 and 104 are Sci-fi
rentals %>% filter(CustomerID == 2,
                   MovieID %in% c(101, 104)) %>%
  count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1     2
```