COMP 120: Lab 7 Worksheet

In this lab, we will begin to explore sources of data that span more than just a single table. In particular, we will examine the frequently encountered relational model of data, where data is broken down and organised into smaller tables that are *related* through shared values in *key columns.* In this lab, we will briefly examine how data is structured and stored in this model, and then examine the functions provided in R to interact with and combine these sources of data.

Remember that anything in a code block should be copied and pasted into the script editor of RStudio and executed so that you can see the desired result for yourself.

Remember to set your working directory and library path (if working on the Student Desktop) at the start of your session. See the Lab 01 for advice on how to do this.

**This lab makes use of several data files, all provided in a single zip file – it is important that you extract these files into your working directory before starting this lab.**

# Overview of the Relational Model

The relational model is a simple but effective means of storing data that promotes accurate and correct information through reduction of duplicated data. The relational model is very frequently encountered in organisations and is arguably the most common source of data that you will encounter. It works by *normalising* data into a series of small tables, each table designed to store closely related records of information. To combine into larger sources of information (those that might be more easily analysed and worked with), the information in these tables are related through *key* columns, which serve the purpose of identifying individual records (primary keys) and identifying the relationships between records in different tables (foreign keys).

Before starting with working with relational data in R, it is a good idea to examine a simple source of relational data in another tool, such as Excel. Open the three files, patients.csv, stats.csv and measurements.csv in Excel, and answering the following questions:

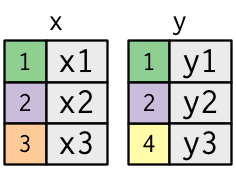
1. What columns are used to relate the patient and measurement tables?
2. What columns are used to relate the stats and measurement tables?
3. What is the height of the patient “Cathryn Bray”? How did you determine this?

# Manipulating Relational Data

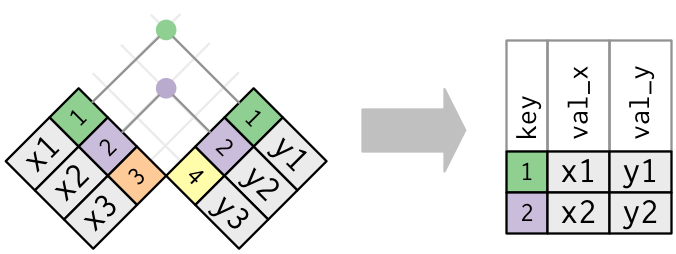
As with previous labs, the functions that you are using here are part of the tidyverse, so we need to load that library:

library(tidyverse)

Data scientists are typically more concerned with querying data, rather than updating, inserting or deleting records. Therefore, we will concentrate here on the functions used to *join records* from separate tables. Joining is the process of matching records from two tables according to their key column values. We are typically interested in two types of join – inner joins (finding the subset of rows with matching values in both tables) and outer joins (keeping all the records from one table, and matching records from a second table where key column values are equal). It is possible to visualise the process of joining data – the following example presented in the lectures (sourced from the [R for Data Science textbook](http://r4ds.had.co.nz/relational-data.html)) demonstrates this. Consider the two tables, linked by their left column (which we’ll refer to as the key column):



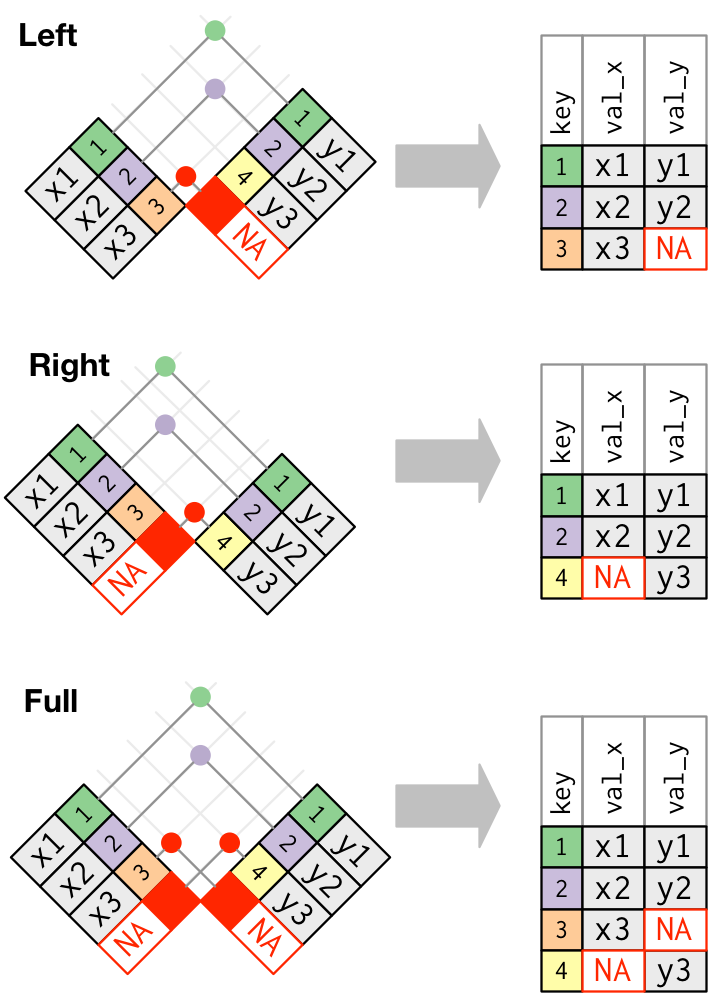
An ***inner*** join between these two tables would find the rows where the key column values match, and return a new table with all the columns from just these rows:



Meanwhile, an ***outer*** join can take on three variants:

* A left outer join, where all the rows of the “left” table are kept, and then the columns of the “right” table are joined to this as appropriate (where the key columns match)
* A right outer join, where all the rows of the “right” table are kept, and then the columns of the “left” table are joined to this as appropriate (where the key columns match)
* A full outer join, where all the rows of both tables are kept, and the related columns from each table are stitched together as indicated by the key column values.

Graphically, these types of outer joins are shown on the next page – note how where there is no related information in the joined table (i.e., where there are no rows with a matching key column value) there is a corresponding NA value in the linked columns:



## Performing Joins in R

Performing joins on two tables in R is fairly straightforward and makes use of the appropriately named inner\_join(), left\_join(), right\_join(), and full\_join() functions. Like the operations from our previous labs, these operators have been designed to be used with the pipe operator (%>%). To demonstrate these functions, we’ll make use of the following tables:

A <- tibble(

key=c("A", "B", "C", "D"),

avalue=c(123, 321, 213, 132)

)

B <- tibble(

key=c("A", "D", "E"),

bvalue=c(987, 789, 911)

)

Applying the inner\_join() function to these two tables produces the following result:

A %>% inner\_join(B)

Joining, by = "key"

# A tibble: 2 x 3

key avalue bvalue

<chr> <dbl> <dbl>

1 A 123 987

2 D 132 789

Notice that only the “common” rows (where the key value matched on both sides of the join) are returned. The left\_join() function performs a slightly different join:

A %>% left\_join(B)

Joining, by = "key"

# A tibble: 4 x 3

key avalue bvalue

<chr> <dbl> <dbl>

1 A 123 987

2 B 321 NA

3 C 213 NA

4 D 132 789

Using the pipe operator makes it clear that table A (which appears on the left hand side of the pipe operator) is the “left” side of the join. Here, the rows with key values B and C are kept, even though there are no matching rows on the right hand side of the join. The right\_join() function performs the opposite of the left join (see result below). It keeps all rows on the “right” side of the join (i.e., table B). From the output below it can be observed that key value E is kept even though there are no matching rows on the left side of the join.

A %>% right\_join(B)

Joining, by = "key"

# A tibble: 3 x 3

key avalue bvalue

<chr> <dbl> <dbl>

1 A 123 987

2 D 132 789

3 E NA 911

Finally, the full\_join() operator is essentially the combination of both a left and right hand join (with duplicates removed):

A %>% full\_join(B)

Joining, by = "key"

# A tibble: 5 x 3

key avalue bvalue

<chr> <dbl> <dbl>

1 A 123 987

2 B 321 NA

3 C 213 NA

4 D 132 789

5 E NA 911

## Combining Joins with Other Functions

The real power of joins becomes apparent when you start to combine them with some of the functions that have been introduced in previous labs, such as *manipulation* (select, filter, mutate and arrange), *aggregation* (group\_by and summarise) and *reshaping* (gather, spread, unite and separate). The previous tables, A and B, are a little too simple to demonstrate more useful operations, so we’ll introduce the following tables:

subject <- tibble(

id=c("1234", "2341", "4321"),

age=c(25, 35, 75)

)

trial <- tibble(

id=c("1234", "4321", "1234", "4321", "1234", "4321",

"1234", "4321", "1234", "4321"),

run=c(1, 1, 2, 2, 3, 3, 4, 4, 5, 5),

reaction=c(97, 137, 106, 138, 103, 152, 102, 148, 111, 153)

)

This setup of the *subject* and *trial* tables might be typical of experimental data (where subject[[1]](#footnote-1) information is kept in one table, and the data collected during an experiment is recorded in a second table called trial). One analysis that we might be interested in is to link the age of the subject with their reaction times, which can be easily done with an inner\_join():

subject %>% inner\_join(trial)

Joining, by = "id"

# A tibble: 10 x 4

id age run reaction

<chr> <dbl> <dbl> <dbl>

1 1234 25 1 97

2 1234 25 2 106

3 1234 25 3 103

4 1234 25 4 102

5 1234 25 5 111

6 4321 75 1 137

7 4321 75 2 138

8 4321 75 3 152

9 4321 75 4 148

10 4321 75 5 153

To make it easier to read a single subject’s reaction times, we might want to convert the table into wide format to have all the reaction times for a given subject on a single row. To obtain this, all we require is a further pipe and call to the spread function that we introduced in the previous lab:

subject %>% inner\_join(trial) %>% spread(run, reaction)

Joining, by = "id"

# A tibble: 2 x 7

id age `1` `2` `3` `4` `5`

<chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>

1 1234 25 97 106 103 102 111

2 4321 75 137 138 152 148 153

Notice that subject 2341 hasn’t appeared in the result. This is because we performed an inner join. We might want to include this subject in the results, to make it clear and obvious that we have no recordings against their record. Given that the subject table appears on the left hand side of the pipe, we can achieve this with a left\_join() call:

subject %>% left\_join(trial)

Joining, by = "id"

# A tibble: 11 x 4

id age run reaction

<chr> <dbl> <dbl> <dbl>

1 1234 25 1 97

2 1234 25 2 106

3 1234 25 3 103

4 1234 25 4 102

5 1234 25 5 111

6 2341 35 NA NA

7 4321 75 1 137

8 4321 75 2 138

9 4321 75 3 152

10 4321 75 4 148

11 4321 75 5 153

Notice that the missing subject is now present, albeit with NA values against its record. Now, we can spread the table as before:

subject %>% left\_join(trial) %>% spread(run, reaction)

Joining, by = "id"

# A tibble: 3 x 8

id age `1` `2` `3` `4` `5` `<NA>`

<chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>

1 1234 25 97 106 103 102 111 NA

2 2341 35 NA NA NA NA NA NA

3 4321 75 137 138 152 148 153 NA

This result has achieved the desired result, although it also produces an undesirable <NA> column (a consequence of the NA value produced in the run column due to the left join). Fortunately, we can fix this with a very simple rearrangement of where the use of the spread() function takes place:

subject %>% left\_join(trial %>% spread(run, reaction))

Joining, by = "id"

# A tibble: 3 x 7

id age `1` `2` `3` `4` `5`

<chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>

1 1234 25 97 106 103 102 111

2 2341 35 NA NA NA NA NA

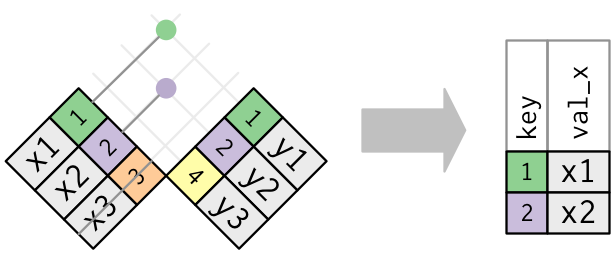
3 4321 75 137 138 152 148 153

Essentially, this means that the “spread” call is done on the trial table prior to being joined to the subject table. This means that the NA values are not produced until the join has completed.

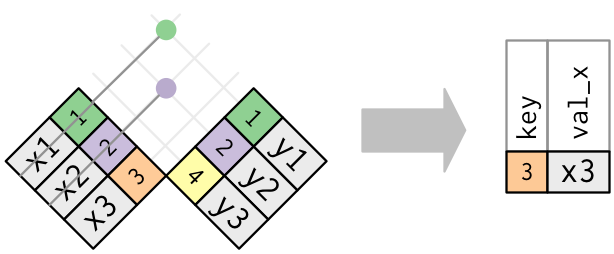
This interaction with the subject and trial tables is a good demonstration of typical use of the join operators that combines data from multiple tables. The joins that we have seen so far are also called *mutating joins* since the result of the join creates more new columns (i.e., by adding new columns to one table from matching observations in another).

Remember, *filtering joins* (semi join and anti join) were also covered in the lectures. These joins filter observations in one table based on whether or not they match an observation in the other table. A *semi\_join* keeps all observations in the left table that have a match in the right table. On the other hand, the *anti\_join* drops all observations in the left table that have a match in the right table (or in other words, it keeps observations in the left table that do not have a match in the right table). Graphical representations of these two joins are given below.

**Semi-join:**



**Anti-join:**



These two filtering joins can be handy in situations such as the ones mentioned in the questions below.

Write code by using the appropriate *filtering join* functions to answer the following questions:

1. Print only those rows from the subject table where a subject has participated in a trial.
2. Print row(s) from the subject table where a subject has NOT participated in a trial.

From here, you should be able to make intelligent use of these operators, combined with the operators from previous labs, to obtain most of the data that you may require for your analysis.

# Mastery Tasks

Yet again, the best way to become comfortable with the join functions presented in this lab is to practice and we’ll do exactly that through the mastery tasks this week. Specifically, you need to perform the following operations on the relational data that we have provided for this lab:

1. Use an appropriate combination of join function introduced in this lab and aggregation functions (see the previous lab!) to manipulate the subject and trial tables to produce the following result:

id age meanreaction

<chr> <dbl> <dbl>

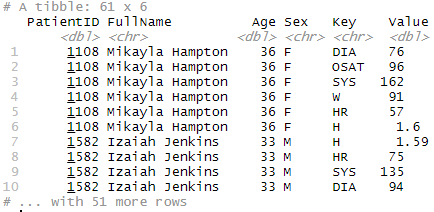
1 1234 25 104.

2 2341 35 NA

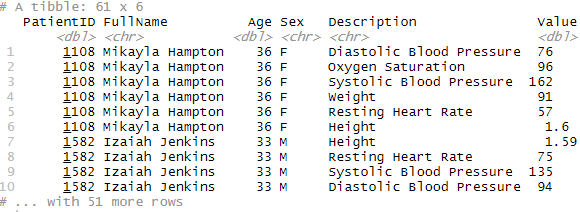
3 4321 75 146.

Note that the meanreaction column is produced by computing the mean value (per subject) of the reaction column. Hint: You can group a dataset based on one or more columns. The exact columns to group\_by will depend upon the problem at hand.

1. Read the patients.csv, stats.csv, and measurements.csv files into tibbles in R. You will use these three tibbles for this task and the remaining tasks. Once the files have been read into tibbles, using an appropriate joining function introduced in this lab, produce the following result which contains 61 rows and six columns. Note, you will just need to join two tables to achieve this result.



1. Now, starting with the code you created for the task above, produce the result shown below. Hint: *Key* column from task 2 has been replaced with *Description* column. For this task, you will need to join the result from the previous task with an appropriate tibble. Also, use appropriate data manipulation functions to achieve the desired result.



1. Examine the contents of the patient-stats.csv file (either by reading it into R, or inspecting it in Excel). Then, use an appropriate combination of joins on these tables, manipulation and/or reshaping functions, to reproduce the result presented in that file (i.e., the patient-stats.csv file). Write the result to a CSV file called widemeasurements.csv. It is OK if the order of rows in your results isn’t exactly the same as the order given in patient-stats.csv file. **To receive full marks for this mastery task, you need to perform all of this operation (after the reading of files) in a single command (i.e., several functions that are chained together using pipes)!** If you are unable to do so, there will be a penalty of 5% (0.15 out of 3).

Put all the required code into a single script that has been given to you (mastery-07.R). Place a short comment before each block of code to indicate what your code does. When you have completed the tasks, **submit your work on Blackboard before 4pm on Thursday, 10th of September**. **Not writing good comments will attract a penalty of 5%**.

1. A subject is a person who participated in an experiment. [↑](#footnote-ref-1)