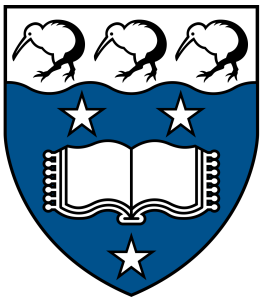
THE UNIVERSITY OF AUCKLAND

MASTERS THESIS

Visualising Data Wrangling Operations: Joins and Reshaping

Author:
Charco HUI

Supervisors:
Anna FERGUSSON
Chris WILD



A thesis submitted in fulfilment of the requirements for the degree of Master of Science

in the

Department of Statistics

June 12, 2019

THE UNIVERSITY OF AUCKLAND

Abstract

Department of Statistics

Master of Science

**Visualising Data Wrangling Operations:
Joins and Reshaping**

by Charco HUI

In the recent update of **iNZight**, a joining and reshaping module was introduced. This module provides a tool for users to join and reshape data sets. Though it is not difficult to apply these operations and obtain the result with a tool like **iNZight**, there is a lack of tutorials or tools that explain the underlying process of these operations in an easy to understand manner. In this report, why these data operations are useful and important will be discussed. Software will also be developed that attempts to teach key concepts associated with joining and reshaping data sets through animations.

Acknowledgements

I would like to acknowledge my supervisors, Anna Fergusson and Chris Wild with my deepest appreciation. Anna always gave me new ideas and Chris was always available when I needed help. It was tough learning a new programming language, but I feel very fortunate that I got introduced into interactive graphics. I had a lot of fun developing this piece of software.

Next, I would like to thank all my friends in the postgraduate lab, they all supported me physically and mentally though out the year.

Lastly, I would like to thank Jot He for helping me with mathematical problems and Michelle He for proof reading.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Data Joins

1.1.1 Background

Over the recent years, technology has been growing rapidly. It is now easier to collect data. Combined with the large volume of data we already have in archives, there are many data sets we can use to perform statistical analysis on. The problem is that even though there are many data sets we can use, it is not common to have one set of data which contains everything we want. One example is regional data. If we wish to do statistical analysis on the all regions, we would need combine multiple data sets together. Most analysis functions in common statistical analysis software only take in one data set.

To solve this problem we can merge the data sets together, in a process that is also known as "data joins". There are many ways we can join these data sets together but it seems like there are not much tutorials that explicate the manipulation of data structures. Therefore, we should focus on how to educate this in an descriptive manner, with this being said we will create a set of animations for a few common data joins.

1.1.2 Current approach of teaching data joins

Text Approach

Understanding how these joining operations work and how to use them can be challenging. Some people can do it by imagining the data structure but those who can't would have a difficult time to learn them just from reading about them.

One explanation found about left joins is that it "returns all rows from the left table, even if there are no matches in the right table". For those who are familiar with joins, this may seem obvious and easy to understand but for those who are not may not understand what "no matches" mean. No matches actually means that there are no identical key columns.

Venn Diagrams (Static)

When it comes to images, Venn diagrams are the most popular way of teaching SQL joins. If we try to search for "SQL join tutorials", we will see Venn diagrams everywhere (See Fig. 1.1).

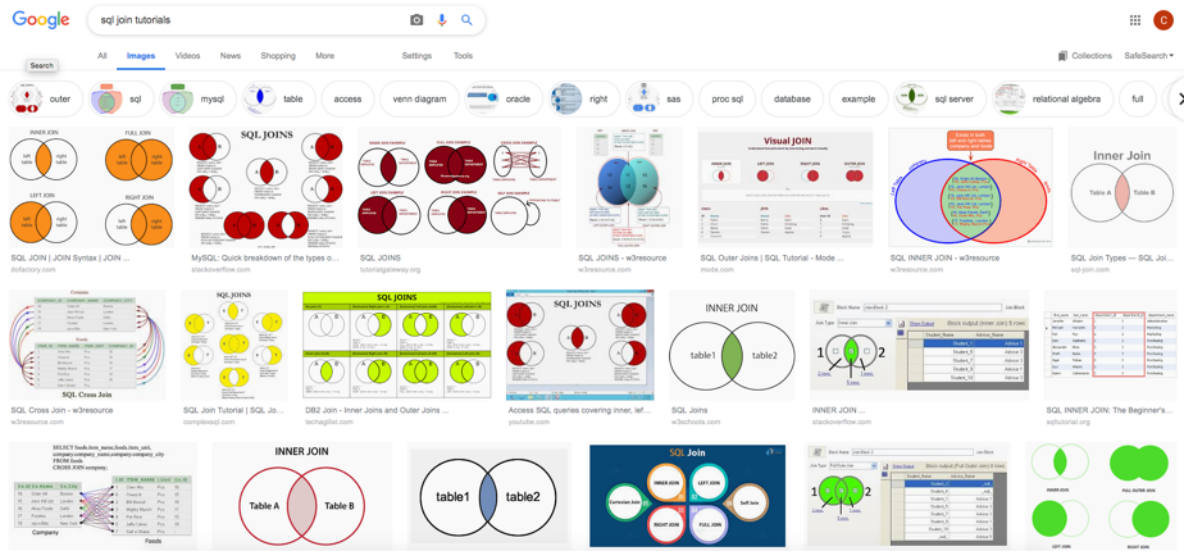


FIGURE 1.1: SQL join tutorials

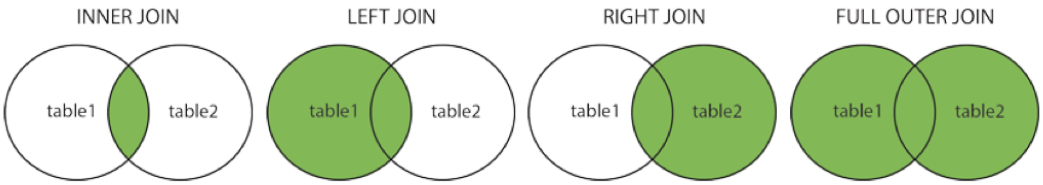


FIGURE 1.2: SQL Join Venn diagrams

Although diagrams like Fig. 1.2 are a popular way of teaching joins, there are many problems with them.

- Venn diagrams are useful in probability because we can think of the diagrams as sets and we can see visually what intersects or unions are. However, when it comes to data joins, the diagrams don't look like data tables.
- They are too vague about how the joins work. For example "inner join" isn't just the intersect of two tables, there is much more to it.
- The multiple type of joins are similar in terms of how the rows from different tables are joined together. The essential differences are in how they treat missing values.

Regardless of the problems, the useful features are,

- The structure of joins are shown. For example, the left join is information from table 1 plus the intersection of table 1 and table 2. Full join is the combination of table 1 and table 2. They are not incorrect, just insufficient.
- These Venn diagrams show us which rows ends up represented in the join but do not tell us anything about how the rows are joined together.
- These diagrams would be a useful visual resource to explain the basic structure of the joins, they are just not suitable to be the only such resource.

1.1.3 R for Data Science Approach

In their influential book R for Data Science, Wickham and Grolemund (2017) have a section explaining different type of joins. They did not use Venn diagrams to convey the idea of joins, however, they did use Venn diagrams for other purposes, so this omission is significant.

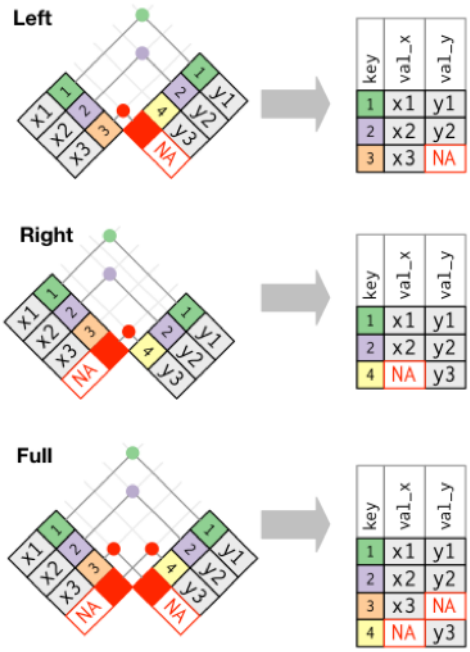


FIGURE 1.3: Data Join Diagrams in R for Data Science

Instead, they show two separate tables as in Fig. 1.3, where each row is coloured differently. Then, there lines which link the rows together indicating that these rows are joined together. At the end there is a result table.

The advantages of this approach compared to Venn diagrams.

- Each table and rows in all tables are shown nicely.
- There are line linking rows that are matched together, and also unmatched rows.
- There is a result table showing what the table would look like after being joined.

What is missing from this approach.

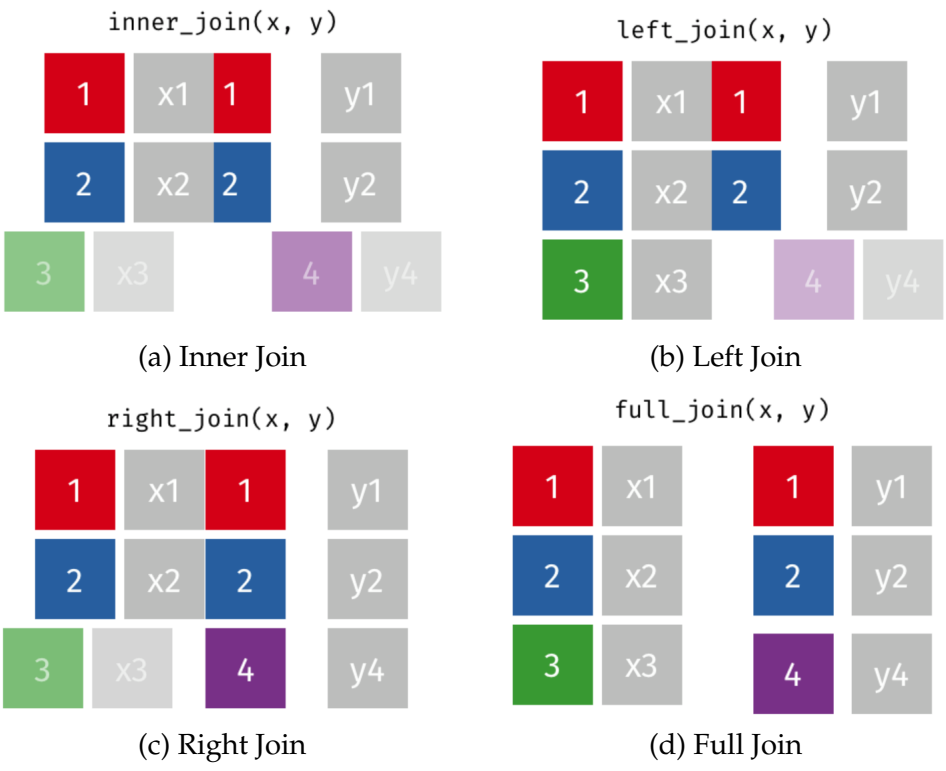
- For those who are not familiar with these joins, it would be difficult for them to understand it without any verbal descriptions.
- They are very cryptic and they do not show effects of multiple matches.
- Matching rows are shown by having the same colour when there more rows, eventually unique colours will run out.
- There are no messages showing why the linked rows are joined together or why some rows disappears after being joined.

With the technology we have today, the ultimate goal would be to use animations to convey the idea of joins.

Dynamic Approach

Static images have been used as the main approach of teaching data joins for a long time, the limitation and lack of features has been explained in the previous sections. So, a new yet better approach should be used, a dynamic approach would be more appropriate. There are many advantages of using animations for teaching purposes. The main one is that humans are more attracted to moving things. They can also focus on them for longer. "This behavioural pattern is rooted deeply in our animal instincts as well as our ancestors' need for self-preservation. In order to keep us safe, our brains have a built-in mechanism that keeps us aware of everything happening around us" [1].

Recently, a set of data joining animations called **tidyexplain** (Aden-Buie and Smith, 2018) on Github (see Fig. 1.4), were created with the **gganimate** (Pedersen and Robinson, 2019) R package.



(a) Inner Join (b) Left Join

(c) Right Join (d) Full Join

FIGURE 1.4: Joining animations in **tidyexplain**

These animations are really similar to the images from the R for data science book, so most of the limitations still remains. The animation lack explanations of how unmatched rows are treated. Although the limitations remain, the advantage of these animations is that the users are given a better picture of how matched rows are joined together.

[1] https://blog.gallereplay.com/static-vs-dynamic-the-psychology-behind-cinemagraphs/

In the next section, we will talk about data reshaping which is also an important part of data pre-processing.

1.2 Data Reshaping

1.2.1 Background

A popular quote *"often 80% to 90% of a data analysis project is spent in making the data reliable enough that the results can be trusted"* (Johnson and Dasu, 2003). This is true, because often we do not receive the data set in the form we want, they are often messy and may take time to transform them into a desired tidy form. It can also be difficult to reshape the data set into the shape that we want. Basically we can break the data pre-handling process into two parts, data cleaning and data tidying. We will focus on the latter one, since data tidying is an important aspect. Therefore, we should educate this with a visualisation where we can allow users to see the process of data reshaping. With this being said, we will create a set of animation for two aspects this.

1.2.2 Tidy data

The idea of the tidy data is that every row is an observation and every column is a variable as shown in Fig. 1.5, this shape of data is used the most in statistical analysis software.

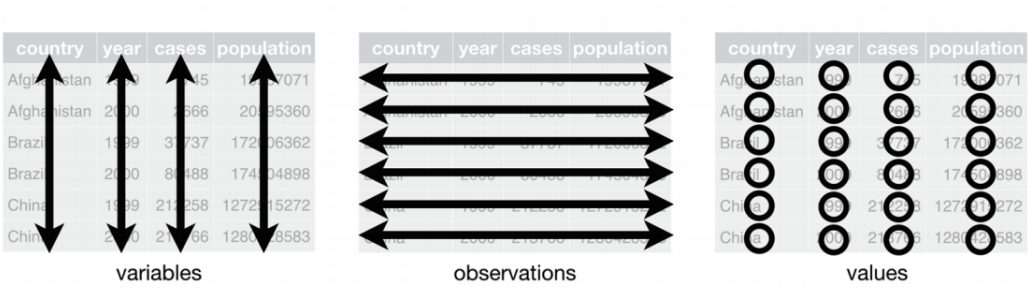


FIGURE 1.5: Tidy data (Wickham and Grolemund, 2017)

However, different software or different functions within a software may take in data sets in various shapes. Below we have the same data set in two shapes, a wide one at Table 1.2 and a long one at Table 1.1.

patient	sex	treatment_num	value
1	M	treatment1	13.90
2	F	treatment1	90.40
3	F	treatment1	15.50
4	M	treatment1	12.70
1	M	treatment2	42.25
2	F	treatment2	16.13
3	F	treatment2	48.84
4	M	treatment2	19.36
1	M	treatment3	98.60
2	F	treatment3	48.04
3	F	treatment3	33.43
4	M	treatment3	4.86

TABLE 1.1: Long data set

patient	sex	treatment1	treatment2	treatment3
1	M	13.90	42.25	98.60
2	F	90.40	16.13	48.04
3	F	15.50	48.84	33.43
4	M	12.70	19.36	4.86

TABLE 1.2: Wide data set

For example, if we want to fit some common models like a regression, we would want the data to be in the wide form, because we want every variable to be in its own column. However, if we want to do a mixed model then we would want the data in a long form. Another advantage of the long data set is that it allows us to produce some plots which we can separate by groups like a box plot by treatment, because we will need a column as an indicator. Therefore, it is important for users to truly understand the underlying process of how reshaping from long to wide or wide to long work. In the next section we will discuss briefly how we can do this in R.

1.2.3 Reshaping in R

In R, to transform the data from wide to long, we can simply use the `gather()` function from the **tidyr** package.

```
> data(toyda_wide)
> data_long = toyda_wide %>% gather(Subject, Score,
    English, Maths)
> data_long
# Name Subject Score
# Alex English  92.2
#  Ben English  63.7
#  Sam English  75.6
# Alex   Maths  88.8
#  Ben   Maths  10.0
#  Sam   Maths  52.9
```

The `key` argument allows us to define the new column we wish create in the new table to store old columns from the original table. For example, we are trying to store columns English and Maths from the wide data into one column named Subject in the new table.

The `value` argument allows us to define the new column to store any values which are in the original data, these are the original values under columns English and Maths

The rest of the parameters are the columns we wish to transform on.

Similarly, for wide to long we can use the `spread()` function, we can think of this as an inverse of the long to wide transformation

```
> data(toyda_long)
> data_wide = toyda_long %>% spread(Subject, Score)
> data_wide
# Name English Maths
# Alex    92.2  88.8
#  Ben    63.7  10.0
#  Sam    75.6  52.9
```

The `key` argument lets us define which column we wish to spread into multiple columns, here, we are trying to spread the Subject column into multiple columns English and Maths.

The `value` lets us define which column contains that corresponding values of the key column (we provided this in the previous argument). These values will then be transformed into the multiple columns that we created.

1.2.4 Current approach of teaching data reshaping

R for Data Science Approach

Again, in the influential book R for Data Science there is a chapter explaining tidy data. Their approach is rather simple, they used a static image with dull grey scale colouring and arrows pointing where each elements from the old table should go into the new table.

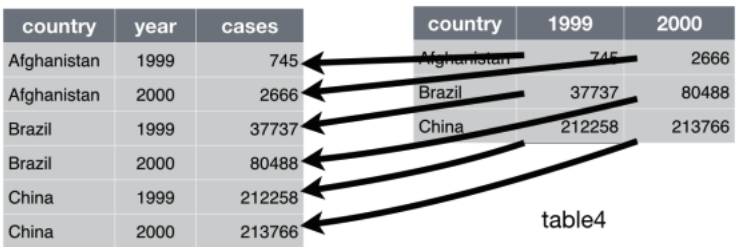


FIGURE 1.6: Wide to Long, R for Data Science

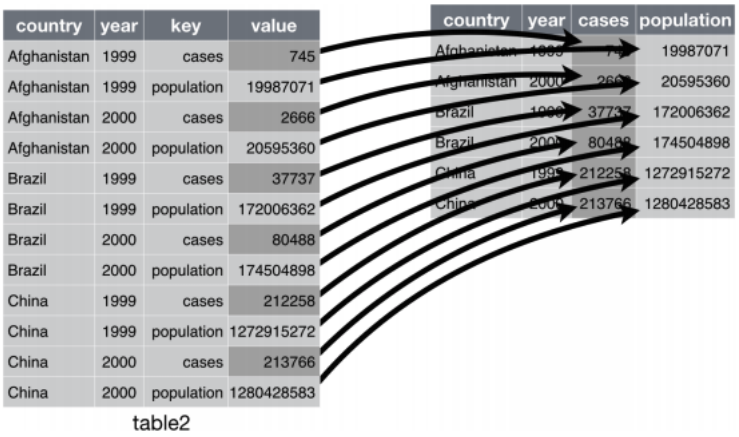


FIGURE 1.7: Long to Wide, R for Data Science

There are not much advantages of this approach except for giving the user a basic idea of how the transformation works.

However, the limitations are.

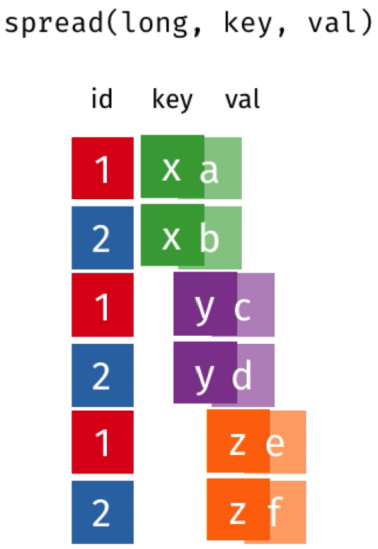
- Not obvious to the user what the images are trying to explain.
- Does not annotate how "1999" and "2000" are used.
- Arrows overprinting on the table makes the table difficult to read.
- Lack of annotations of what the steps are.

Dynamic Approach

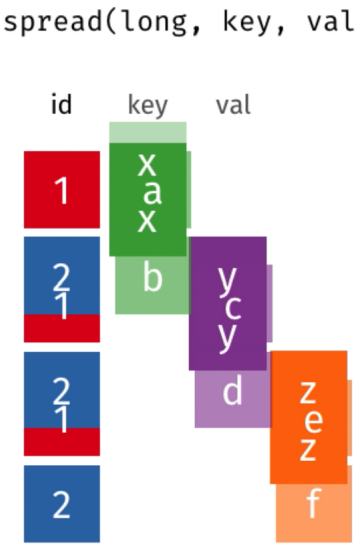
As explained before in the previous sections, a set of data joining animations called **tidyexplain** on Github (see Fig. 1.4), were created with the **gganimate** R package. They also have another set of animations for data reshaping (See Fig. 1.8 and Fig. 1.9).

The advantages and disadvantages of this approach are the same as with the joins. Animations can be easier to understand than static images because they can show more details. This set of animation lack annotations explaining each step and does not use vocabulary that is familiar to learners. They show the process without any explanation. More over, reshaping a data set is actually more complex than data joins because the shape of the data set changes.

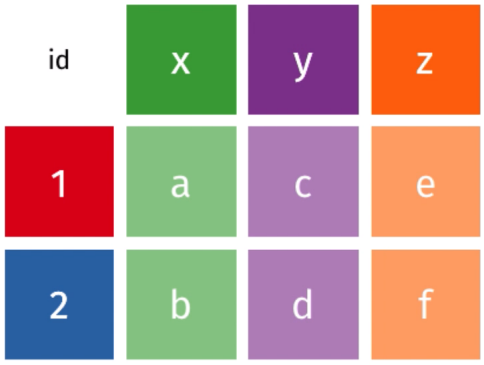
(a) Long to Wide step 1



(b) Long to Wide step 2

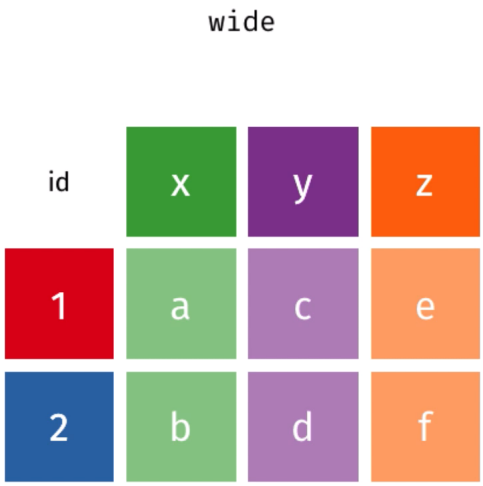


(c) Long to Wide step 3

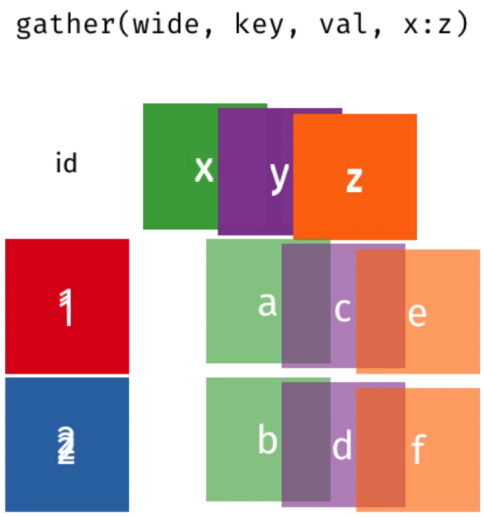


(d) Long to Wide step 4

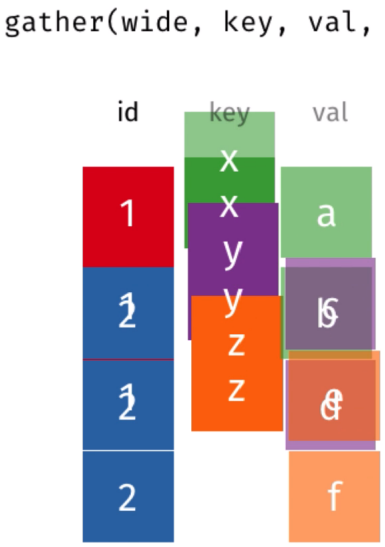
FIGURE 1.8: Long to Wide in **tidyexplain**



(a) Wide to Long step 1



(b) Wide to Long step 2



(c) Wide to Long step 3

(d) Wide to Long step 4

FIGURE 1.9: Wide to Long in **tidyexplain**

1.3 Motivation

The **tidyexplain** approach is a good start but there is more room for improvement. In general to make this more intuitive, we would like,

- To allow users to input their own data to produce the animations.
- Displaying messages to show how the logic behind each step.
- Control of playback speed.
- Ability to save an animation for use in web pages and presentations.

For the data joining module, we would like,

- To allow for more advanced joins.
- Show why or how rows are joined together.
- Show how duplicated rows are handled.
- Make the key column more stand out since it is the foundation of every join.

For the data joining module, we would like,

- To show how or why certain elements are moved.
- Show how elements within a table are related to each column or rows.
- Convey the idea of key and value.

Chapter 2

Methodology for Data Joins

2.1 Software Structure

Based on the shortcomings of the different techniques we discussed in the previous sections, we will try to develop a tool that will not only attack the biggest limitation we have found (intuitiveness), but this tool will also be user friendly. This will allow users who have no experience in data joins to create animations which are easy to understand by just clicking a few buttons, and also provide a support for those who are currently teaching joins.

To create the animations, two programming languages were used, R and JavaScript. HTML and CSS were also used but they are considered to be a markup language and a style sheet language. R was used to process the data, display the animations and provide an interactive dashboard. JavaScript was used to create the animations. Lastly, **htmlwidgets** (Vaidyanathan et al., 2018) is a tool which allows R and JavaScript to communicate with each other and **shiny** (Chang et al., 2019) is a tool used to create the interactive dashboard.

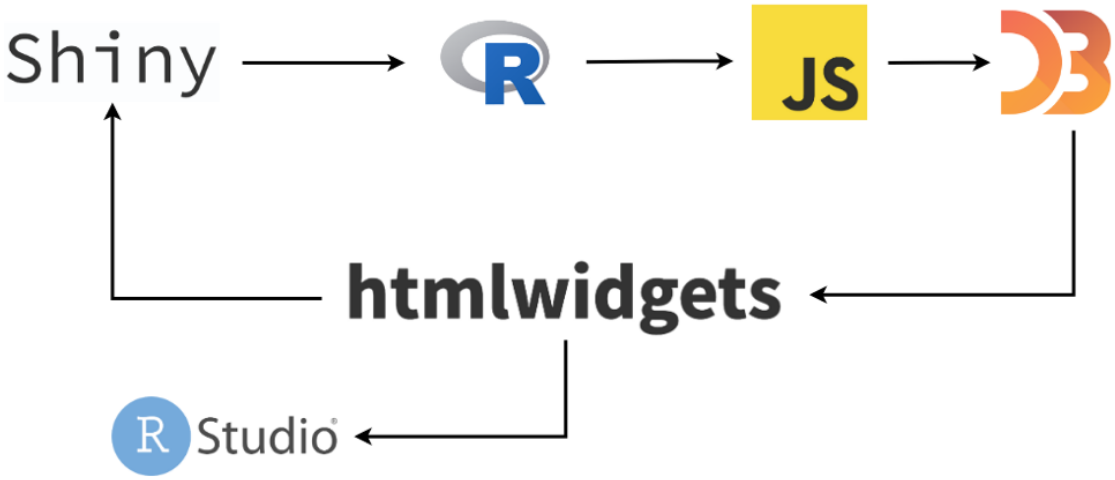


FIGURE 2.1: Software Structure

We are creating dynamic visualisations that can be delivered over the internet that are made using our **shiny** app, and that take user instructions and display the resulting dynamic graphics. The graphics are created using JavaScript, and in particular the **D3** library which provides very useful tools for creating new free-form graphics to display in a browser. An R function called `join_anim` function from our **dataAnim** package takes the user instructions communicated via the **shiny** dashboard, reads in the data sets to be joined, pre-processes the data, and then compiles

a set of instructions to be fed to our JavaScript program. The JavaScript program creates the graphics which are then passed back to **shiny** for display. The graphics can be saved as a (dynamic) **HTML** file that can be given away and used independently of the system.

The first step of the animation is to process the data in **R** by using **dplyr** (Wickham et al., 2019), a package for data manipulation. It provides useful functions to both manipulate and join the data sets. Next this information will be passed to JavaScript, where we use **D3**, a powerful graphing library for creating animations. After the JavaScript creates the animation in a SVG format, we then use **htmlwidgets** in R to wrap the SVG up in a container which lets us display them in a browser or the view panel in **Rstudio**. What **htmlwidgets** also provides is to render the animation into an object that **shiny** can take in. This is useful because we are creating our interactive dashboard in Shiny.

With this tool, users will be able to create animations in **Rstudio** by using the console, or in **shiny** by using the user interface. On top of that, users will be able to save animations they create, based on their own data sets, in a **HTML** format. This is useful because they will be able to share these animations with other people.

2.2 R

Mainly, two packages were used in R, **dplyr** and **tidyr** (Wickham and Henry, 2019). Both packages were used to manipulate and transform data sets provided by the user, so we can extract information shown below to pass into JavaScript.

What happens is when the user input the data sets, R processes it and produces a list of step by step instructions of how the animations are to be created for JavaScript. Some examples are, column one from Table 1 and column two from Table 2 are the key columns. Row one from Table 1 matches row three from Table 2.

Other information passed to JavaScript from R includes:

- The dimensions and information of the two input tables, and the joined table
- The size of the tables in the animation
- The order and the corresponding matching rows throughout the joining animation
- The text of the instructional messages to be displayed when they are required
- How to deal with unmatched rows for different kind of joins

2.3 New package - dataAnim

To produce the joining animations, the `join_anim` function in our **dataAnim** package is used. The compulsory arguments are the type of join to perform, the two tables and the key column to join by. Optional arguments include the speed of the animation and a logical value to choose whether to display the instructional messages. An example of this function is shown in Fig. 2.2.

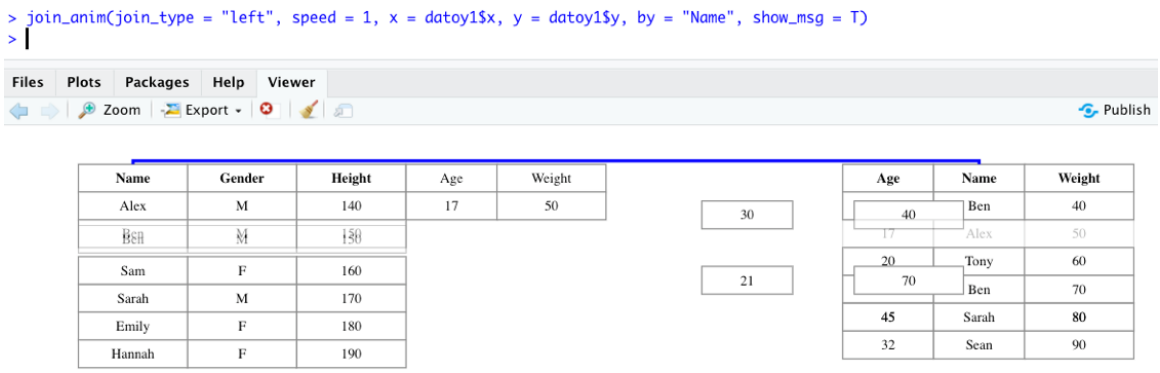


FIGURE 2.2: `join_anim` example

Currently, it supports three type of joins, *left join*, *inner join* and *complete join*. A *right join* is basically the same as a *left join*, except that it just joins onto the right table instead of the left table. *Left join* is also known as *left outer join*, and a full join is also known as a *complete join* or *full outer join*.

The `join_anim` function returns a **htmlwidgets** object which the user can use via the `savewidget` function from **htmlwidget** to save the object as a **HTML** file. The help file for The `join_anim` function follows in Fig. 2.3.

Discussion of the arguments:

- `join_type` allow users to choose the type of join they want perform.
- `x` and `y` are the tables that are being joined together.
- `by` is the key column which defines the join.
- `speed` allows us to give the user control of the speed of the animation.
- `width` and `height` are the size of the animation frame in pixels
- `show_msg` gives the user an option to include explanatory annotations in the animation.

Joining Animation

Description

Function to create joining animations.

Usage

```
function(join_type = "left", x, y, by, speed = 1, width = NULL, height = NULL, show_msg = T)
```

Arguments

join_type	Type of Join.
x	Table x, to pass to join.
y	Table y, to to pass to join.
by	A character value of the variable to join by.
speed	Speed of the animation.
width	Width of the animation frame in pixels.
height	Height of the animation frame in pixels.
show_msg	A logical value indicating whether to show instructional messages in the animation.

Author(s)

Charco Hui

Examples

```
data(datoy1)
join_anim(join_type = "left", speed = 1, x = datoy1$x, y = datoy1$y, by = "Name", show_msg = T)
join_anim(join_type = "inner", speed = 1, x = datoy1$x, y = datoy1$y, by = "Name", show_msg = F)
myanim = join_anim(join_type = "left", speed = 1, x = datoy1$x, y = datoy1$y, by = "Name", show_msg = T)
htmlwidgets::saveWidget(myanim)
```

FIGURE 2.3: `join_anim` help page

2.4 JavaScript - D3

D3 is a popular library in JavaScript to create dynamic interactive data visualisations. **D3** creates graphics in scalable vector graphics (SVG), a file format based on mathematical calculations. This means that the graphics can be zoomed in without becoming pixelated. Many famous and influential companies like the New York Times uses this technology to produce interactive or SVG graphics.

Here, the JavaScript program receives the instructions produced by R and calls **D3** functions to create the animations, then JavaScript sends the animation in SVG format back to R to process the final presentation. The problem with using JavaScript and **D3** is that they run asynchronously, meaning that all the code runs at once, this causes problems because there are many components in our animations that are created during the animation. Therefore when the program is created, we need to be cautious and need to make provisions for the new components that are not displayed in the animation originally. A flow of our JavaScript program is shown in Figure 2.4.

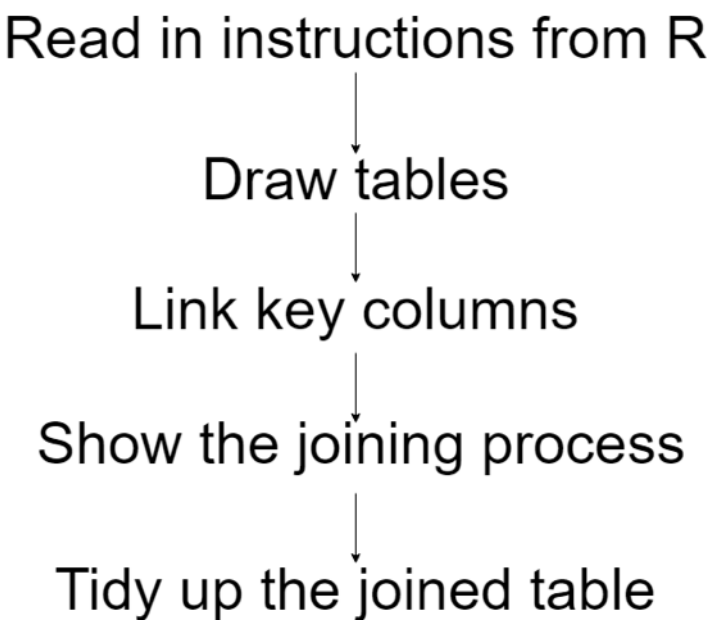


FIGURE 2.4: JavaScript program flow

As shown in the above diagram, the JavaScript program:

1. Reads in the instruction from R and passing R objects directly to JavaScript using the **htmlwidgets** environment.
2. Draws the two tables, at the correct co-ordinates.
3. Implements animation to show which columns are the key columns from each table, and links them together.
4. Starts the joining animation by the type of join the user chose. Instructional messages explaining each steps will also be shown unless the user chooses not to.
5. Tidy up the joined table. At the end the table that we are joining from will fade out, and the joined table will stay. The joined table will then move to the middle indicating that it is the completed table and that the animation is over.

2.5 htmlwidgets and shiny

As discussed at the start of this chapter, the JavaScript program uses **D3** to creates graphics and animations in SVG, which are mostly displayed in internet browsers. Natively, R does not support SVG graphics directly, so, to display the animations in R, we need to use **htmlwidgets**. The **shiny** interactive dashboard for the users, is shown in Fig. 2.5. This is a good extension because this will allow users who have no experience in programming to create the animations by using the command panel displayed in Fig. 2.5.

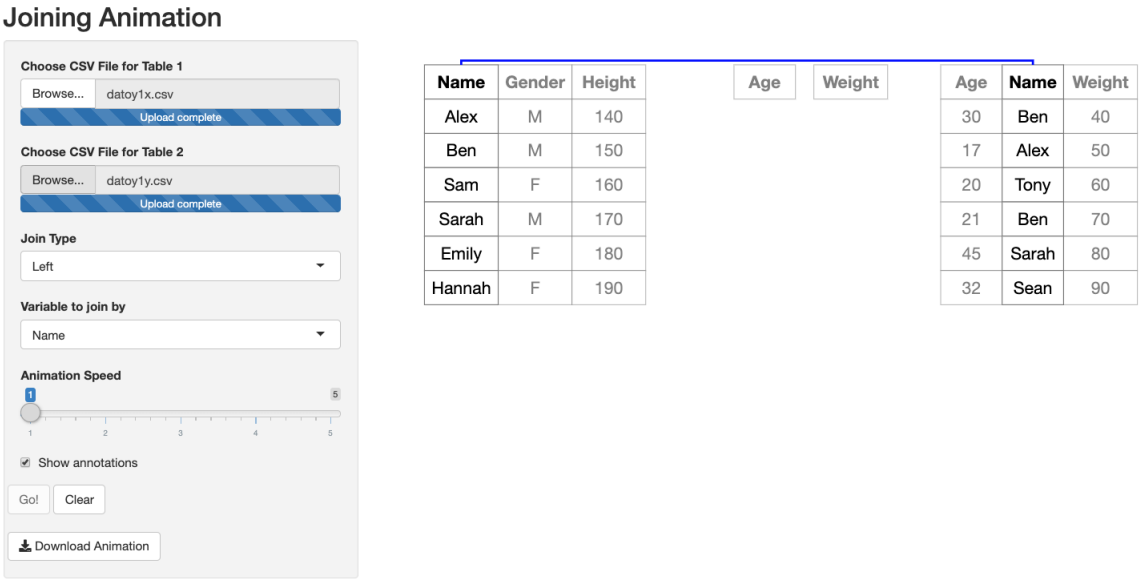


FIGURE 2.5: **shiny** interactive dashboard in **dataAnim**

2.6 Animation

In the previous section, some issues with the existing approach of teaching data joins were discussed. Some improvements were made to address those issues, with the end goal of making these animations as intuitive as possible.

The goals of these animations are to:

- To show the logic of the joins
- Highlight the importance and role of the key variables
- Show the user how each of the rows are being joined together
- Show how the joins handle unmatched rows
- Show how the joins handle multiple matches
- Show how data are moved between tables
- Show how different type of joins handle the tables differently.
- Allow users to download and export the animations.

Chapter 3

Animation for Data Joins

3.1 The logic of joins

Joins allows us to combine the information in different data sets. This could be because we just want to keep all the data together instead of having multiple different sets of data stored separately.

Another reason could be because we have multiple data sets which are related to each other and we would want to combine them together into a single data frame so we can perform statistical analysis on them. Most analysis functions in R take data from a single data frame.

In a sense joins are just trying to combine data sets together based on some conditions, we can think of this as adding more columns or variables to one data set with the new information taken from the other data sets. These conditions are like some sort of instructions we give to the join, so it can output a data set that we desire. In joins, these conditions are defined by the key column, this is discussed in more detail in the next section.

All examples in the following section will be based on the two toy data files shown in Fig. 3.1.

Name	Gender	Height
Alex	M	140
Ben	M	150
Sam	F	160
Sarah	M	170
Emily	F	180
Hannah	F	190

Table 1

Age	Name	Weight
30	Ben	40
17	Alex	50
20	Tony	60
21	Ben	70
45	Sarah	80
32	Sean	90

Table 2

FIGURE 3.1: Toy data set

3.2 Linking key columns

The problem we are trying to solve here is that we want to join the two tables together, to do this we will need to use the key column which is explained above. It gives us information on how the rows from the two tables are joined together. Without the key columns there will be no ways to define what rows belong together.

In this section we will be discussing how to draw attention of the user to the key columns and their role.

Initially, we want the user to focus on the key columns. We do this by first fading out the non-key columns. We use this fading strategy often when we want the user to focus on particular elements at that moment. Second, we show a line linking the key column from both tables. Third, we display a set of words stating which columns we are joining by.

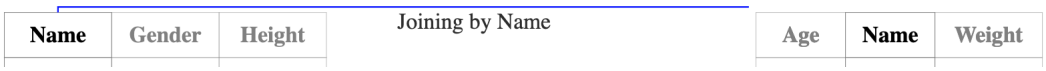


FIGURE 3.2: Key column step 1

Forth, we flash the key column headers on both tables to further focus attention on the key columns.

FIGURE 3.3: Key column step 2

Then we need to show the users the structure of the joined table, to do this, we move all column headers except for the key from Table 2 to Table 1, this create new columns/variables in Table 1. After those elements have been moved over, the user can then see the basic structure of the resulted joined table because the rows below the new columns will be empty as shown in Fig. 3.4. The next step is to fill in the empty parts of the rows.

FIGURE 3.4: Key column step 3

Once this structure has been emphasised the faded-out elements then revert back to normal. The original columns in table 1 will remain bold-ed whereas the new columns are not, this is to remind the user which columns are being joined on.

Name	**Gender**	**Height**	Age	Weight
Alex	M	140		
Ben	M	150		
Sam	F	160		
Sarah	M	170		
Emily	F	180		
Hannah	F	190		

Age	**Name**	**Weight**
30	Ben	40
17	Alex	50
20	Tony	60
21	Ben	70
45	Sarah	80
32	Sean	90

FIGURE 3.5: Key column step 4

Next we will talk about how to fill in the empty rows, the joining process.

3.3 Joining process

In all joins, as part of joining information on rows from the two tables that belong together, elements from the key column in Table 1 will try find a match in the key column of table 2 sequentially until the end of Table 1.

3.3.1 The case of a Single match

Here, Alex is in the first row of Table 1. The join then tries to find Alex in table 2. In this scenario, there are matched elements from both key columns, since we have Alex in both tables, this means that these rows can be joined together.

First, to draw attention to why and how these rows are being joined together, if there is a match on both tables, both matched elements from both key columns flash, and a line then links them together (Fig. 3.6).

FIGURE 3.6: Single match step 1

Second, a message will show explaining that we are currently "Matching Alex" (Fig. 3.7).

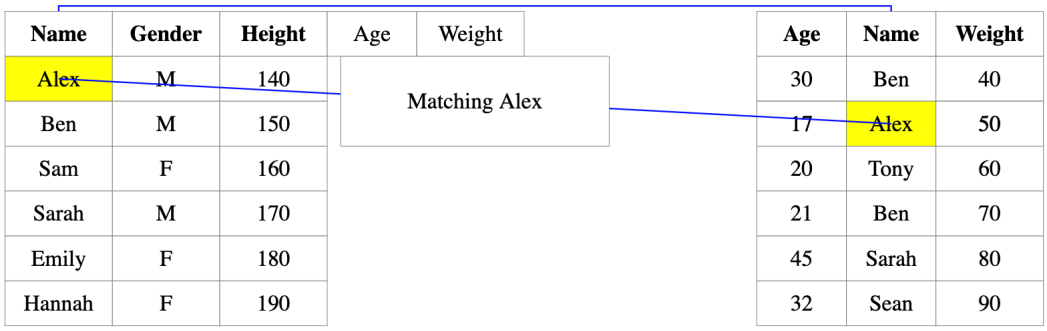


FIGURE 3.7: Single match step 2

Then the actual join will perform, all elements except the key from the matched row in table 2 will move across to table 1 (Fig. 3.8 caught mid-move).

Name	Gender	Height	Age	Weight
Alex	M	140		
Ben	M	150		
Sam	F	160		
Sarah	M	170		
Emily	F	180		
Hannah	F	190		

17 50

Age	Name	Weight
30	Ben	40
17	Alex	50
20	Tony	60
21	Ben	70
45	Sarah	80
32	Sean	90

FIGURE 3.8: Single match step 3

Lastly, to show the user that the information on Alex in Table 2 has now been used, and we don't need to play attention to that particular row anymore, we fade out that particular row in Table 2 (Fig. 3.9).

Name	Gender	Height	Age	Weight
Alex	M	140	17	50
Ben	M	150		
Sam	F	160		
Sarah	M	170		
Emily	F	180		
Hannah	F	190		

Age	Name	Weight
30	Ben	40
17	Alex	50
20	Tony	60
21	Ben	70
45	Sarah	80
32	Sean	90

FIGURE 3.9: Single match step 4

We then move on to Ben, then Sam and so on down Table 1.

3.3.2 The case of Multiple matches

This occurs when a row in Table 1 matches multiple rows in Table 2.

Like the single match scenario, to allow the users focus on the rows that are being matched, their elements will flash, but this time we will see that there is more than one line matching the rows because there is more than one match found.

To reinforce the user that there are more than one rows that match and we want to draw attention to some different behaviour, we show "2 matches found for Ben" (Fig. 3.10).

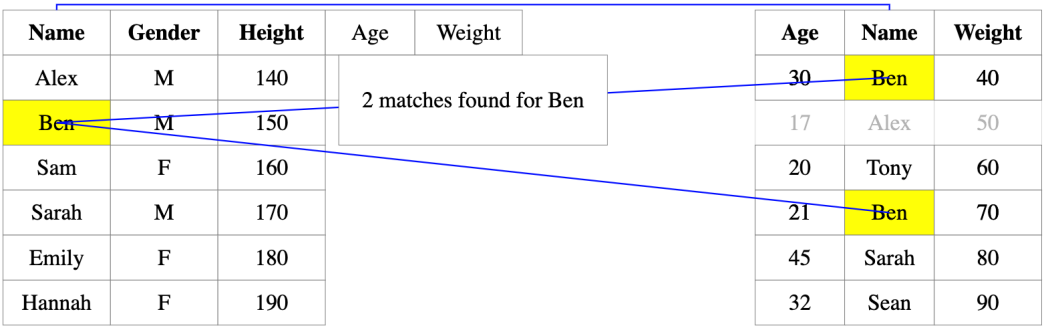


FIGURE 3.10: Multiple matches step 1

We want to move two rows of data from Table 2 across to Table 1 This causes a problem, because there is only one row in Table 1 which matched Table 2.

To solve this we need to compensate by adding an extra row to Table 1, duplicating the matched row from Table 1 (Fig. 3.11). Fig. 3.11 also shows the two "Ben" rows from Table 2 in the process of being moved across.

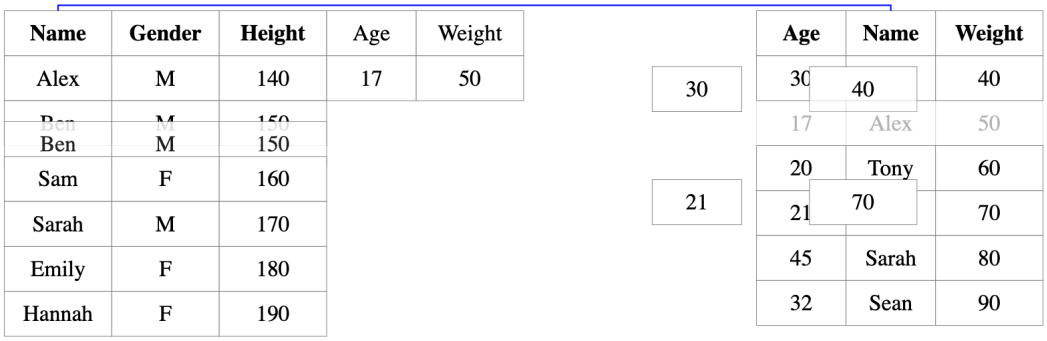


FIGURE 3.11: Multiple matches step 2

3.3.3 The case of No match

What happens when we get to a row in Table 1 with no match in Table 2? Here we don't have Sam in the key column of Table 2 (Fig. 3.12).

To show this we first flash the row that is currently looking for a match to show the user that this row is currently looking for a match. Second, we animate a line to move from that particular element towards Table 2 to show that the join is in the process of looking for a match. Third, we stop the line from moving towards table 2 and a pop up question mark. Lastly, we pop up a message telling the user that there were no match found.

Actions for the no-matches found . case differs between different type of join, as discussed in in paragraphs that follow.

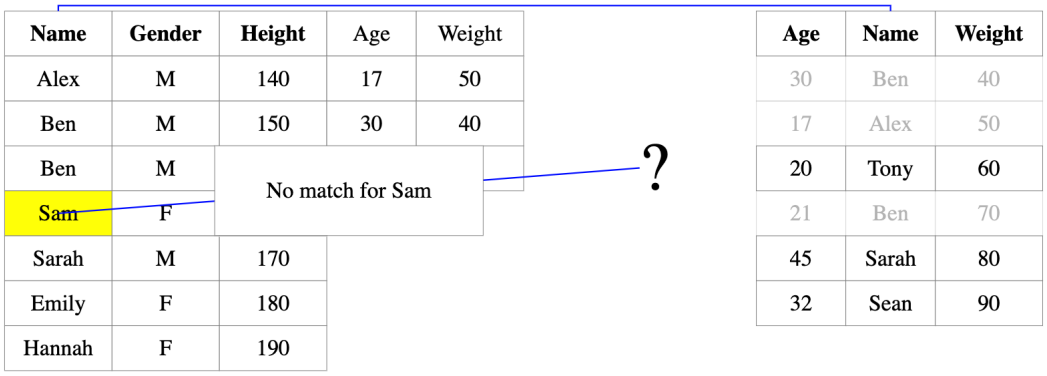


FIGURE 3.12: No match

3.4 Left/Right Join

We only show a left join because we get the same resulting join as a right join by left-joining in the reverse order. *Left/Right joins* are also known as *left/right outer joins*.

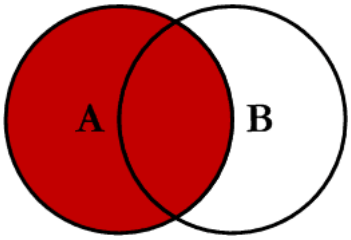


FIGURE 3.13: Venn diagram of Left Join

The definition of a left join is that it returns data relating to all rows from Table 1, and the matching rows from Table 2.

For a left join, when there are no matches found, the row in Table 1 is still kept, but there is no information on Table 2 variables for this unit. Therefore those values from the new Table 2 variables are missing (NA). To show this we then replace the missing values with NA which is the term for missing value in R. Since Sam was not found in Table 2, we fill their Age and Weight columns with NA.

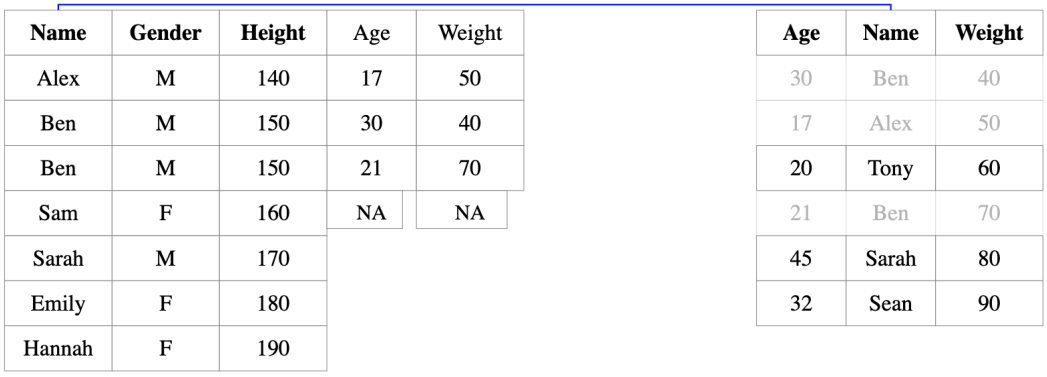


FIGURE 3.14: Left Join step 1

After the last row in Table 1 finishes matching, the join is complete. To show the user which rows were used, we can see that some rows are faded out in Table 2. By

this strategy, all the usable information in Table 2 has already been used. What is left in Table 2 corresponds to units that are not to be used because they do not appear in Table 1 (see Fig. 3.15).

Name	Gender	Height	Age	Weight
Alex	M	140	17	50
Ben	M	150	30	40
Ben	M	150	21	70
Sam	F	160	NA	NA
Sarah	M	170	45	80
Emily	F	180	NA	NA
Hannah	F	190	NA	NA

Age	Name	Weight
30	Ben	40
17	Alex	50
20	Tony	60
21	Ben	70
45	Sarah	80
32	Sean	90

FIGURE 3.15: Left Join step 2

To show the user that the join is complete, Table 2 fade away because it is not needed anymore, and Table 1 is moved to the middle so the user can see clearly the resulted joined Table (Fig. 3.16).

Name	Gender	Height	Age	Weight
Alex	M	140	17	50
Ben	M	150	30	40
Ben	M	150	21	70
Sam	F	160	NA	NA
Sarah	M	170	45	80
Emily	F	180	NA	NA
Hannah	F	190	NA	NA

FIGURE 3.16: Left Join step 3

3.5 Inner Join

The main idea of a inner join is to combine information on only those units that appear in **both** Table 1 and Table 2. That is why they are typically represented by a intersection in a Venn diagram.

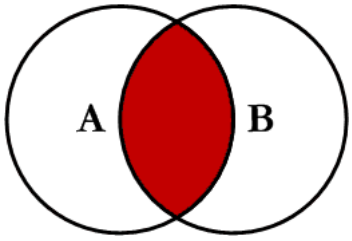


FIGURE 3.17: Venn diagram of Inner Join

The definition of a inner join is it returns rows that have matching key column values in both tables. This is similar to the left join explained above but since it only returns rows that have matching key columns in both tables. When no match is found for a Table 1 row in Table 2, we remove that row in Table 1 instead of filling in NA for the missing values (Fig. 3.18, 3.19).

Name	Gender	Height	Age	Weight
Alex	M	140	17	50
Ben	M	150	30	40
Ben	M	150	21	70
Sam	F	160		
Sarah	M	170		
Emily	F	180		
Hannah	F	190		

Age	Name	Weight
30	Ben	40
17	Alex	50
20	Tony	60
21	Ben	70
45	Sarah	80
32	Sean	90

FIGURE 3.18: Inner Join step 1

At the end of the join, the user will be able to see which rows did not find a match, from the gaps between rows in Table 1. This is to remind the user that there are rows that were removed because there were no matches for them.

Name	Gender	Height	Age	Weight
Alex	M	140	17	50
Ben	M	150	30	40
Ben	M	150	21	70
Sarah	M	170	45	80

Age	Name	Weight
30	Ben	40
17	Alex	50
20	Tony	60
21	Ben	70
45	Sarah	80
32	Sean	90

FIGURE 3.19: Inner Join step 2

Similar to left join, to show that the join is finished, Table 2 will be removed and the resulted joined table will be centred. The only difference is that all the rows will be pushed together to fill in the gaps produced by the removed rows (Fig. 3.20).

Name	Gender	Height	Age	Weight
Alex	M	140	17	50
Ben	M	150	30	40
Ben	M	150	21	70
Sarah	M	170	45	80

FIGURE 3.20: Inner Join step 3

3.6 Full Join

The full join is also known as a complete join. The main idea of a full join is to combine information on all units that appear in either Table 1 or Table 2 or both. That is why they are often represented by a union in a Venn diagram.

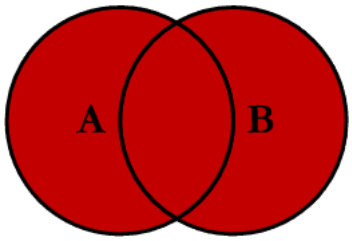


FIGURE 3.21: Venn diagram of Full Join

The definition of a full join is it returns all rows relating to key column values found in Table 1, Table 2 or both. For a complete join, the first step of the joining process is the same as a left join. The difference is that when the matching process is done, the left join is complete, whereas the full join now will move the unmatched rows from the Table 2 to Table 1.

The problem here is that we will need to move the remaining unmatched rows from Table 2 that have not been used, these are rows in Table 2 that are not faded out (because they have already been used).

First, to catch the users attention, we flash the elements from the key column in the unmatched rows in Table 2. Second, we animate a line and a question mark showing the user that they are the unmatched rows and there is no where for them to go (Fig. 3.22).

Name	Gender	Height	Age	Weight
Alex	M	140	17	50
Ben	M	150	30	40
Ben	M	150	21	70
Sam	F	160	NA	NA
Sarah	M	170	45	80
Emily	F	180	NA	NA
Hannah	F	190	NA	NA

?

Age	Name	Weight
30	Ben	40
17	Alex	50
20	Tony	60
21	Ben	70
45	Sarah	80
32	Sean	90

FIGURE 3.22: Full Join step 1

Third, we show a message indicating that (Fig. 3.23). Then we move the unused rows across to Table 1, this is to remind the user what the next step is.

Name	**Gender**	**Height**	Age	Weight
Alex	M	140	17	50
Ben	M	150	30	40
Ben	M	150	21	70
Sam	F	160	NA	NA
Sarah	M	170	45	80
Emily	F	180	NA	NA
Hannah	F	190	NA	NA

Move across unused rows

Age	**Name**	**Weight**
30	Ben	40
17	Alex	50
20	Tony	60
21	Ben	70
45	Sarah	80
32	Sean	90

FIGURE 3.23: Full Join step 2

Fourth, we move the unmatched rows from Table 2 across to Table 1, to their corresponding column in row order (Fig. 3.24).

Name	**Gender**	**Height**	Age	Weight
Alex	M	140	17	50
Ben	M	150	30	40
Ben	M	150	21	70
Sam	F	160	NA	NA
Sarah	M	170	45	80
Emily	F	180	Tony	NA
Hannah	F	190	NA	NA
			Sean	

Age	**Name**	**Weight**
30	Ben	40
17	Alex	50
20	Tony	60
21	Ben	70
45	Sarah	80
32	Sean	90

20 60

32 90

FIGURE 3.24: Full Join step 3

After the rows are moved across, the cells corresponding to the Table 1 variables will be empty. To complete the join we will need to fill these with NAs to indicate that these are missing values.

Therefore, fifth, to convey this idea, we flash this region in red, question marks will also appear showing the user that this region is missing, and that we do not have any information in these data (Fig. 3.25).

Name	Gender	Height	Age	Weight
Alex	M	140	17	50
Ben	M	150	30	40
Ben	M	150	21	70
Sam	F	160	NA	NA
Sarah	M	170	45	80
Emily	F	180	NA	NA
Hannah	F	190	NA	NA
Tony	?	?	20	60
Sean			32	90

Age	Name	Weight
30	Ben	40
17	Alex	50
20	Tony	60
21	Ben	70
45	Sarah	80
32	Sean	90

FIGURE 3.25: Full Join step 4

Then we replace these empty elements with NA, similar to what we did with a left join (Fig. 3.26).

Name	Gender	Height	Age	Weight
Alex	M	140	17	50
Ben	M	150	30	40
Ben	M	150	21	70
Sam	F	160	NA	NA
Sarah	M	170	45	80
Emily	F	180	NA	NA
Hannah	F	190	NA	NA
Tony	NA	NA	20	60
Sean	NA	NA	32	90

Age	Name	Weight
30	Ben	40
17	Alex	50
20	Tony	60
21	Ben	70
45	Sarah	80
32	Sean	90

FIGURE 3.26: Full Join step 5

Lastly, at the end of the join, we isolate the resulted joined table to the middle to show the user that this is finished (Fig. 3.27).

Name	Gender	Height	Age	Weight
Alex	M	140	17	50
Ben	M	150	30	40
Ben	M	150	21	70
Sam	F	160	NA	NA
Sarah	M	170	45	80
Emily	F	180	NA	NA
Hannah	F	190	NA	NA
Tony	NA	NA	20	60
Sean	NA	NA	32	90

FIGURE 3.27: Full Join step 6

Chapter 4

Methodology for Data Reshaping

4.1 Software Structre

The software structure for the reshaping module is the same as the joining module, refer to Section 2.1 and Fig. 2.1.

4.2 R

In the reshaping module, we still use the **dplyr** and **tidyr** package. The difference is that in the joining module, we used **dplyr** more frequently because it provided us with functions to join data sets together. In the reshaping module, **tidyr** was used more frequently because it contains functions to reshape data sets.

The idea is the same, but instead of passing information to JavaScript to perform joining animations, we pass in information to perform reshaping animations. They include,

- The dimensions and information of the inputted table.
- The dimension and information of the resulted table.
- The text of the instructional messages to be displayed.
- The name and column number of the key columns.
- The name and column number of the values columns.
- The corresponding position of where each cell elements within a table goes (in matrix notation).

4.3 dataAnim

The reshaping module brings two new functions to the **dataAnim** package, the `spread_anim` function to generate a long to wide animation and the gather_anim function generates a wide to long animation.

The compulsory arguments include the speed of the animation, the data set to apply transformation on, the name of the key and the name of the value. `gather_anim` requires an addition argument `col`, this indicates the columns we wish to reshape on. Below is the help page for `spread_anim` shown in Fig. 4.1 and `gather_anim` shown in Fig. 4.2.

spread_anim {dataAnim} R Documentation

Spread Animation

Description

Long to Wide transformation.

Usage

```
spread_anim(key, value, data, speed = 1, width = NULL, height = NULL)
```

Arguments

key	Column used to spread out to multiple columns.
value	Column containing values of the key.
data	Data to pass into the function.
speed	Speed of the animation.
width	Width of the animation frame in pixels.
height	Height of the animation frame in pixels.

Author(s)

Charco Hui

Examples

```
data(toyda_long)
spread_anim(key = "Subject", value = "Score", data = toyda_long)
myanim = spread_anim(key = "Subject", value = "Score", data = toyda_long)
htmlwidgets::saveWidget(myanim, file = "myanim.html")
```

FIGURE 4.1: `spread_anim` help page

gather_anim {dataAnim} R Documentation

Gather Animation

Description

Wide to long transformation.

Usage

```
gather_anim(key, value, data, col, speed = 1, width = NULL,
  height = NULL)
```

Arguments

key	New column name to contain old columns names.
value	New column name to contain old column valuess.
data	Data to pass into the function.
col	Columns in the original data to transform on.
speed	Speed of the animation.
width	Width of the animation frame in pixels.
height	Height of the animation frame in pixels.

Author(s)

Charco Hui

Examples

```
data(toyda_wide)
gather_anim(key = "Subject", value = "Score", col = c("English", "Maths"), data = toyda_wide)
myanim = gather_anim(key = "Subject", value = "Score", col = c("English", "Maths"), data = toyda_wide)
htmlwidgets::saveWidget(myanim, file = "myanim.html")
```

FIGURE 4.2: `gather_anim` help page

Discussion of the common arguments:

- `data` is the table to perform reshaping on.
- `speed` allows us to give the user control of the speed of the animation.
- `width` and `height` are the size of the animation frame in pixels.

Discussion of the `spread_anim` arguments:

- `key` is the column the user chooses to spread out to multiple columns.
- `value` is the column the that contains the values of the key columns.

Discussion of the `gather_anim` arguments:

- `key` is the new column name the user wishes to use to contain the old column names.
- `value` is the new column name the user wishes to use to store those values.
- `col` are the columns the user wishes to reshape on.

To produce these animation, one method is to use **Rstudio**. An example of generating a wide to long animation using the `gather_anim` in **Rstudio** is shown in Fig. 4.3.

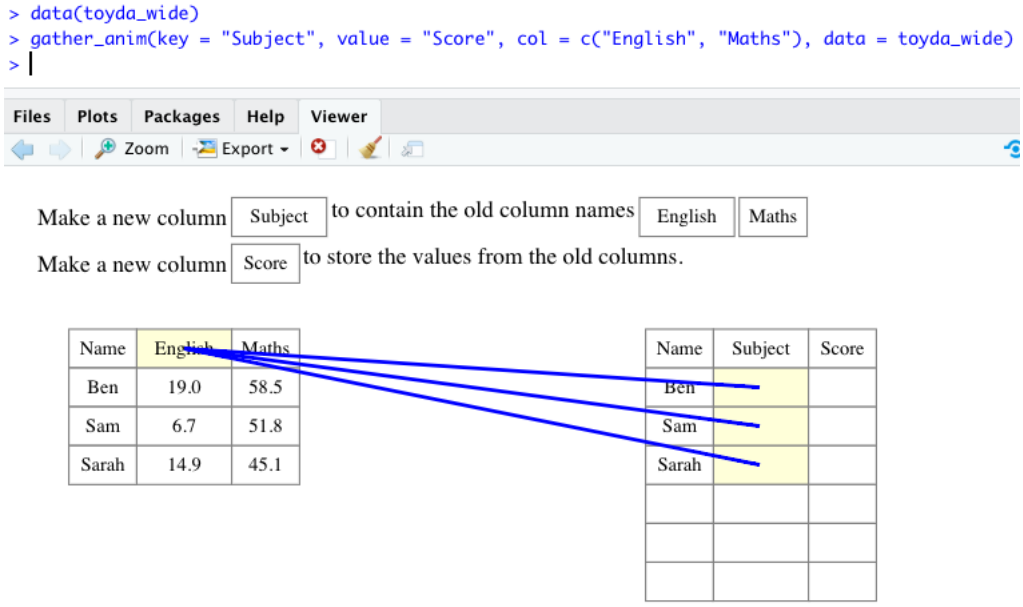


FIGURE 4.3: `gather_anim` Example

Another method to generate these animations is to use our **shiny** interactive dashboard, we will discuss this in Section 4.5.

4.4 JavaScript - D3

In the reshaping module, the use of JavaScript remains the same. However the flow is different. As shown in Fig. 4.4, the JavaScript program:

1. Reads in the instruction from R and passing R objects directly to JavaScript using the **htmlwidgets** environment.
2. Draw the original table given by the user.
3. Give a brief introduction of the what the animation will be based on.
4. Display the structure of the reshaped table. This table will be empty, it is shown to give the user an idea of what the resulted table will look like.
5. Starts the reshaping animation and display the instructional messages when necessary.

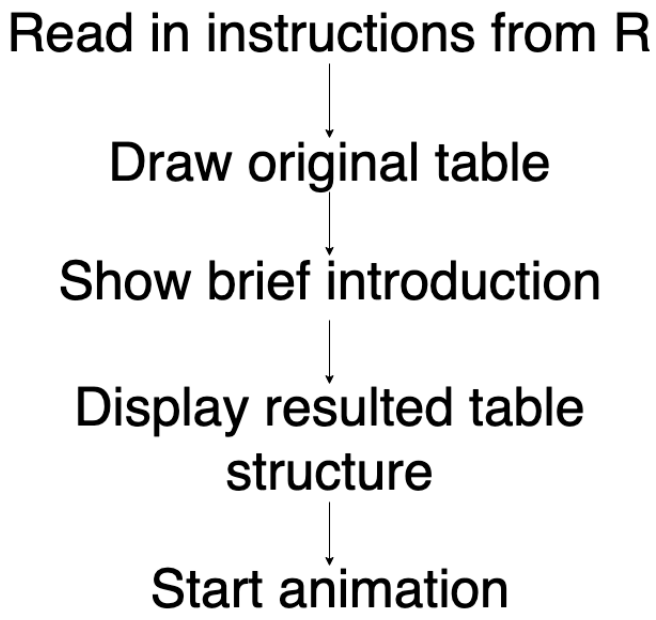
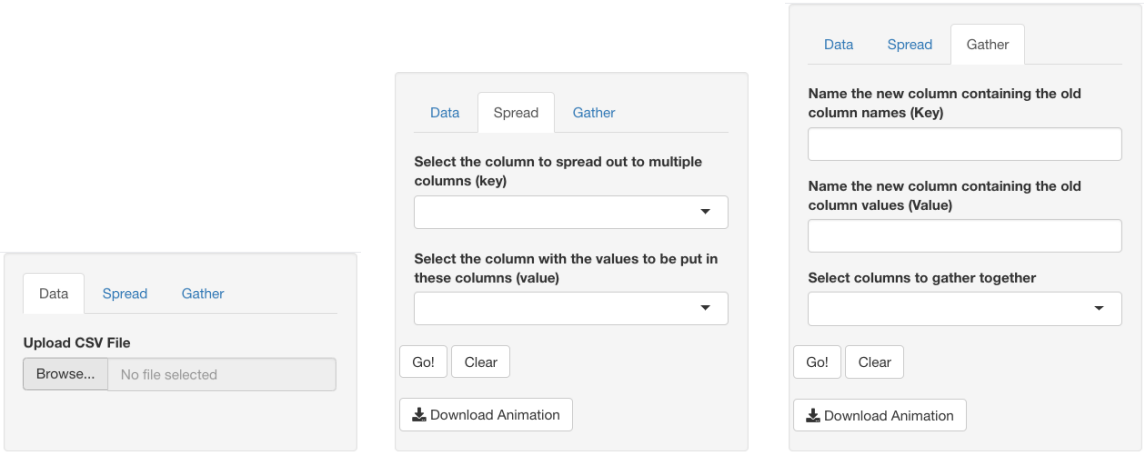


FIGURE 4.4: JavaScript program flow

4.5 htmlwidgets and shiny

As shown in Fig. 4.5, the shiny interactive dashboard for the reshaping module is different to the joining module. There are three tabs, Tab 1 allow the users to upload their data set in a CSV format, Tab 2 allow users to generate a long to wide animation (Spread) and Tab 3 allow users to generate a wide to long animation (Gather).



(a) Tab 1 (b) Tab 2 (c) Tab 3

FIGURE 4.5: **shiny** interactive dashboard (reshape module)

4.6 Animation

In the previous section, some issues with the existing approach of teaching data reshaping were discussed. Some improvements were made to address those issues, with the end goal of making these animations as intuitive as possible.

The goals of these animations are to:

- To show the logic of the data reshaping
- Highlight the importance and role of the key and value (columns)
- Show the user the relationship between variables in the original table and the resulted table.
- Show the underlying process of a wide to long transformation (Spread)
- Show the underlying process of a long to wide transformation (Gather)
- Show how data are moved between tables
- Allow users to download and export the animations.

Chapter 5

Animation for Data Reshaping

In this chapter we will discuss how we use animations to expression data reshaping.

5.1 Long to Wide (Spread)

5.1.1 Logic of Long to Wide

The logic of the long to wide transformation in the `spread` function of **tidyr** is to use the `key` and `value` column. Note that the `key` does not have the same definition as the key column in data joins. The `key` here is the column the user wishes to use to spread out to multiple columns and the `value` is the column that contains their values.

5.1.2 Long to Wide Animation

Before the animation starts, we tell the user that we will be using a old column from the original table given by the user to create new columns in the resulted table by using a message. Here we will use the old column Subject to spread out (create) two columns English and Maths.

We then move elements English and Maths from the original table up to the message because we want the user to know that these are the columns we will be using during the animation (Fig. 5.1). We use a table cell like design for this word within the sentence because we want the user to know that we are referring to a certain element in a particular table. We will use this table cell technique often through out the reshaping animation.

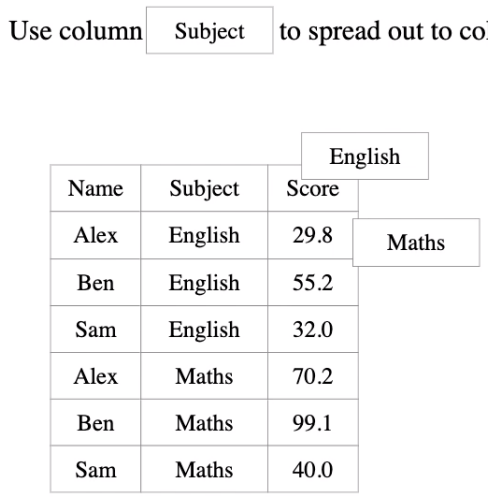


FIGURE 5.1: Long to Wide step 1

Second, we show another message informing the user that we will use the values in the Score column for this transformation (Fig. 5.2). Again, the word Score is in a table cell like design.

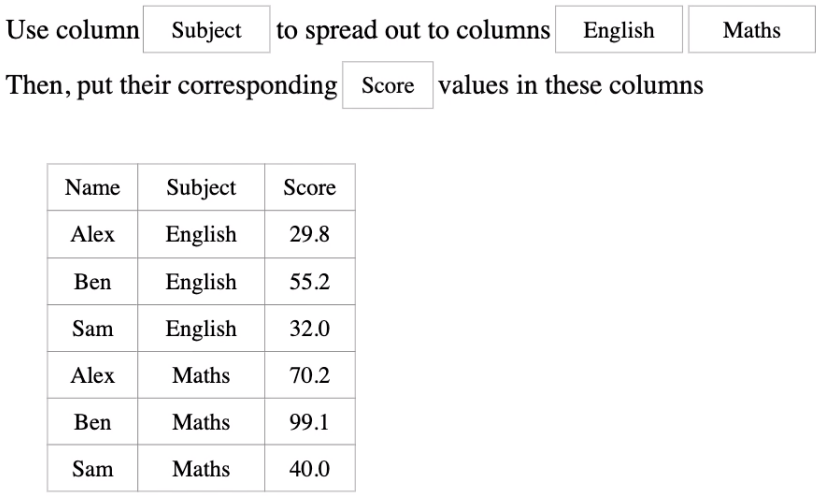


FIGURE 5.2: Long to Wide step 2

Third, we show the structure of the resulted table. This is to give the user an idea of what the resulted table will look like. This table is currently empty and only contain column names, these empty cells will be filled in though out the animation (Fig. 5.3).

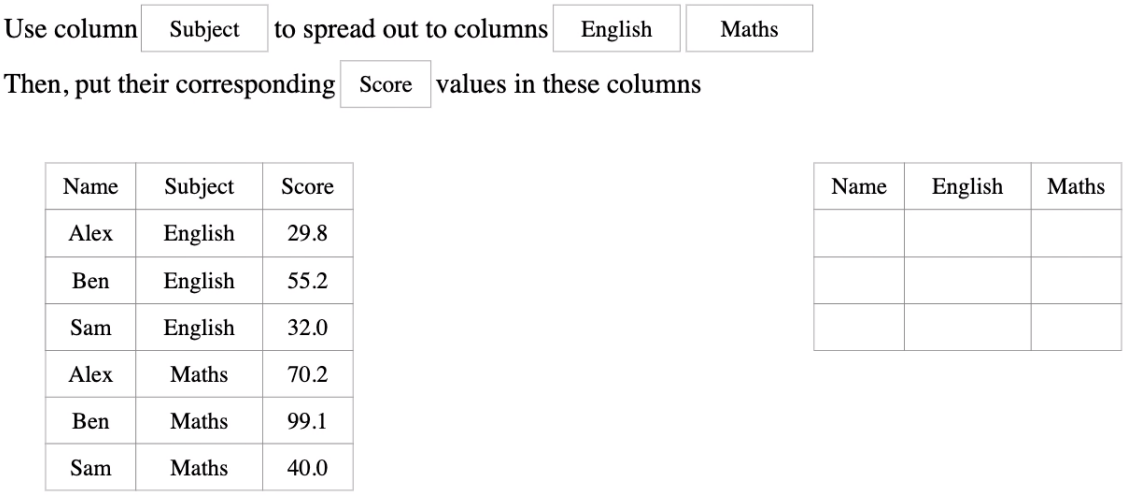


FIGURE 5.3: Long to Wide step 3

Fourth, we fade out the tables and the text in the background. Then we show a message informing the user that the animation will now start and that the first step is to move across unique values from the column that is neither the key or value across to the resulted table (Fig. 5.4). For example, here we show a message "Move across unique values from Name". We fade out the background to allow the user to focus on this message.

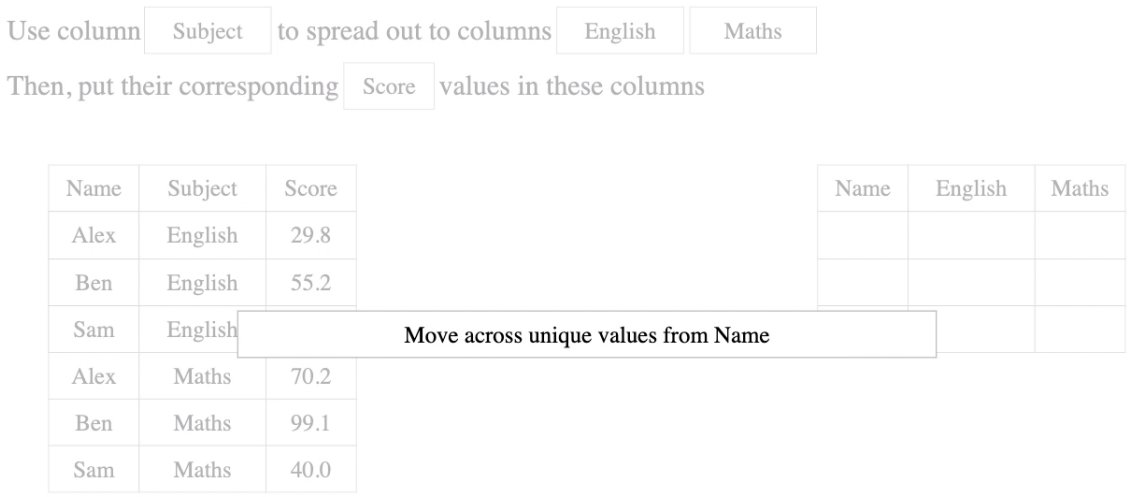


FIGURE 5.4: Long to Wide step 4

Fifth, we flash the unique elements in the Name column, to catch the users attention Fig. 5.5).

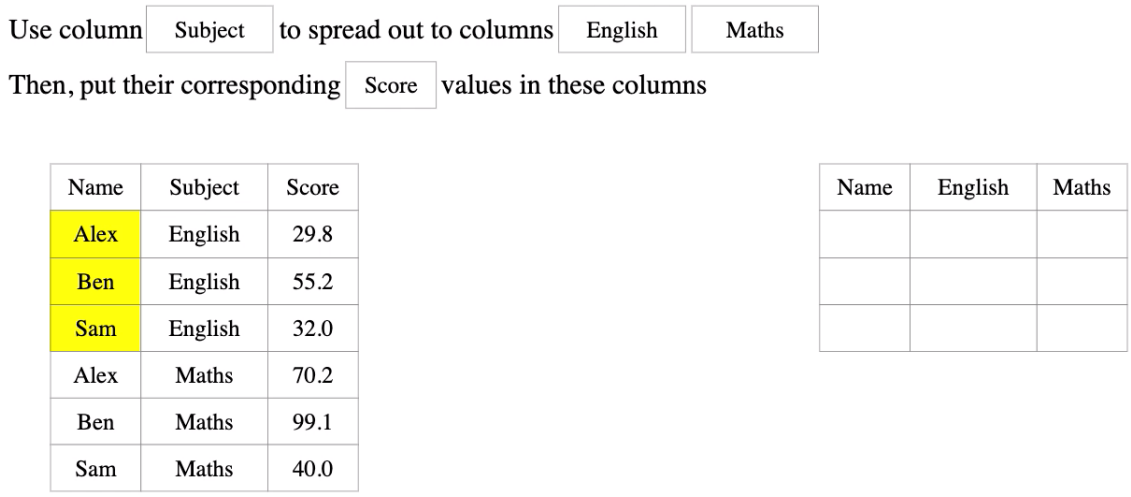


FIGURE 5.5: Long to Wide step 5

Sixth, we move the unique names over to the resulted table Fig. 5.6, caught mid-move).

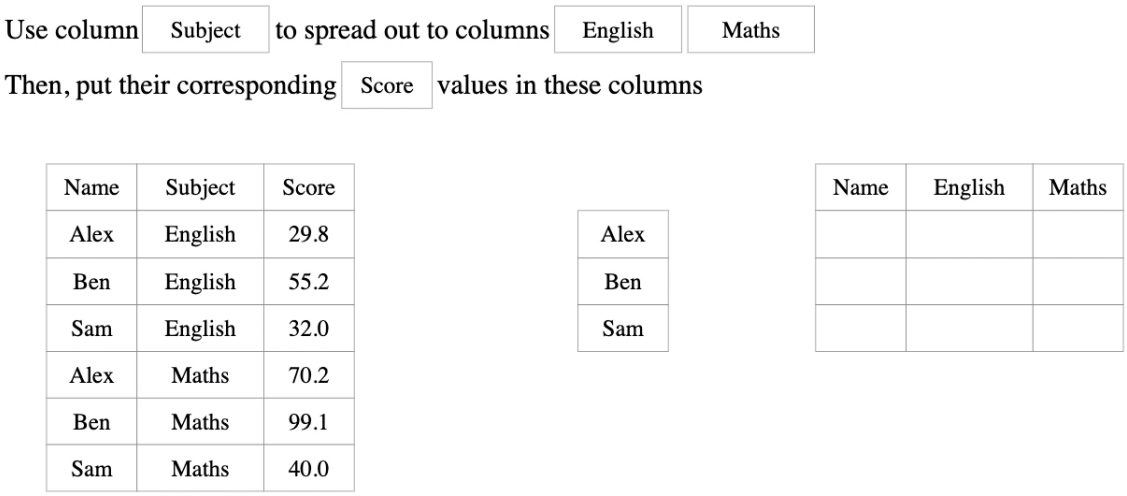


FIGURE 5.6: Long to Wide step 6

Next, we flash the first set of unique row in the original table. Here, it is Alex and English in the original table. To catch the users attention to this row, the elements containing these values in the resulted table will also flash (Fig. 5.7). Here, we want the user to understand the relationship between the two tables.

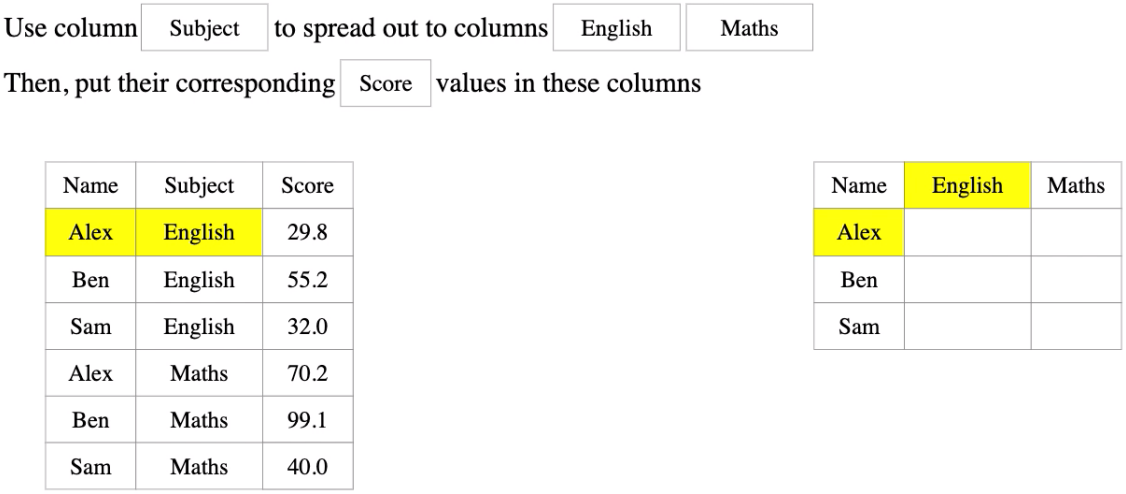


FIGURE 5.7: Long to Wide step 7

The values of the previously flashed rows will then move into its corresponding location in the resulted table. The correct location for this cell will be the intersect of the two flashed cells in the result table (Fig. 5.8, caught mid-move). The animation continues sequentially down the rows of the original table.

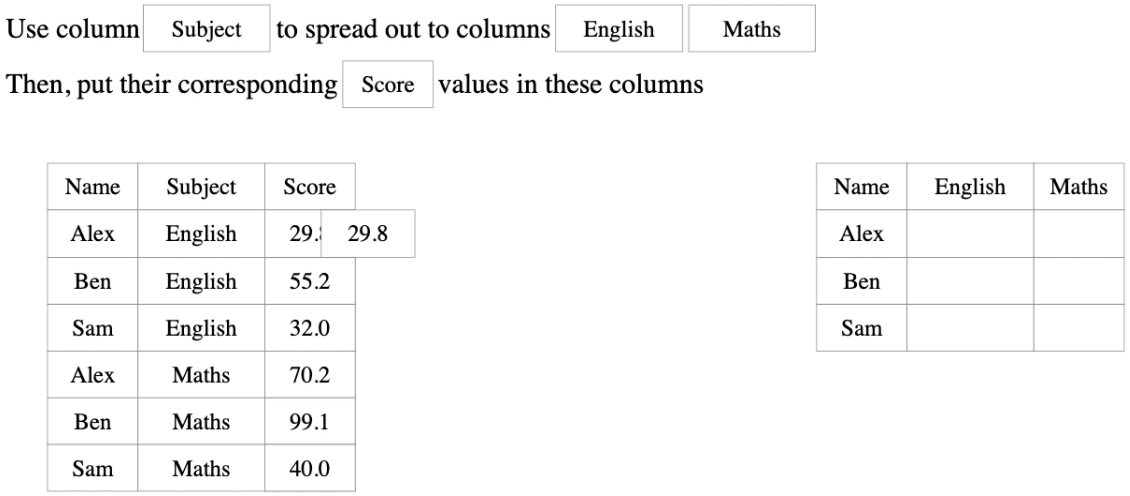


FIGURE 5.8: Long to Wide step 8

We then do the same for Maths after we are finished with English (Fig. 5.9).

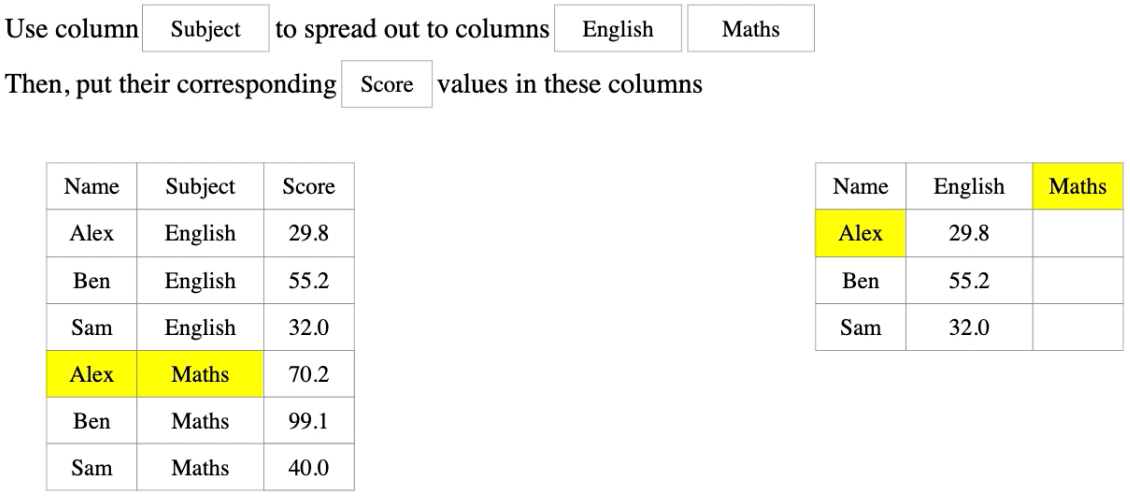


FIGURE 5.9: Long to Wide step 9

Lastly, when the animation finishes, the animation will stop (Fig. 5.10).

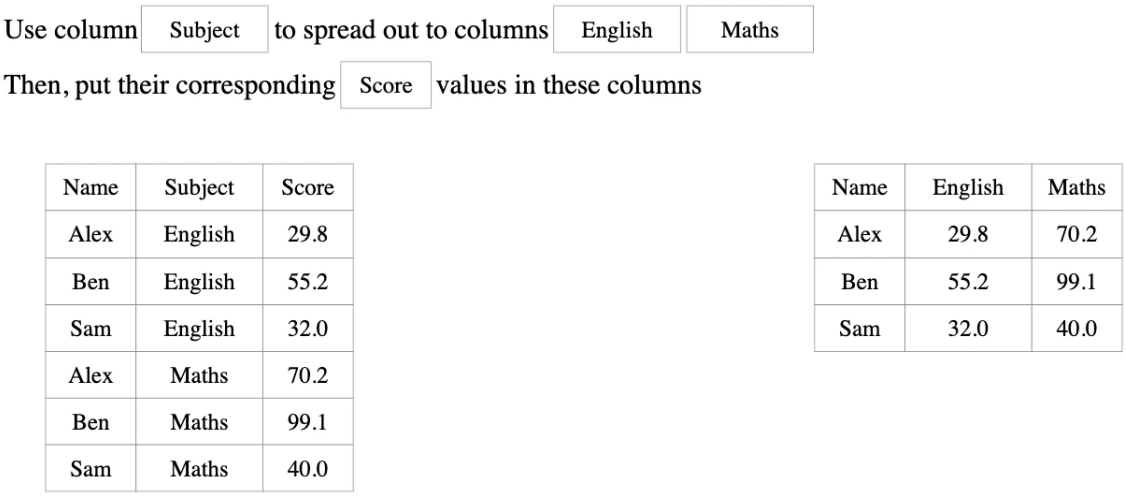


FIGURE 5.10: Long to Wide step 10

5.2 Wide to Long (Gather)

5.2.1 Logic of Wide to Long

The logic of the wide to long transformation in the `gather` function of **tidyr** is to use the `key` and `value` column, this is the same as `gather`. However, the meaning of the `key` and `value` is different. Here, `key` is the new column name the user wishes to use to contain old column names. The `value` is the new column name to contain the old column values. An additional argument is required by `gather`, the user will also need to provide the columns they wish to transform the data on.

5.2.2 Wide to Long Animation

Before the animation starts, we tell the user that we will create a new column to contain old column names by using a message. Here we will create a new column named Subject to contain the old column names English and Maths.

We then move elements English and Maths from the original table up to the message because we want the user to know that these are the columns we will be using during the animation (Fig. 5.11).

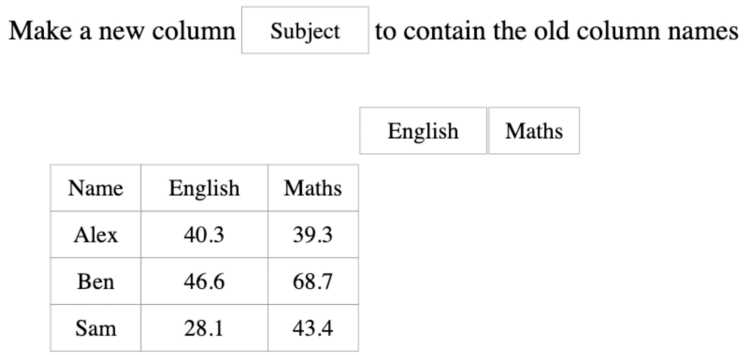
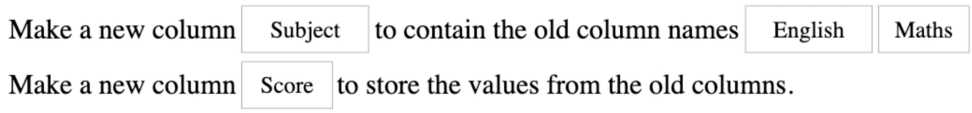


FIGURE 5.11: Wide to Long step 1

Second, we show another message informing the user that we will create another new column to contain the old column values. Here we will create a new column in the resulted table called Score (Fig. 5.12).



Name	English	Maths
Alex	40.3	39.3
Ben	46.6	68.7
Sam	28.1	43.4

FIGURE 5.12: Wide to Long step 2

Third, we show the structure of the resulted table. This is to give the user an idea of what the resulted table will look like. This table is currently empty and only contain column names, these empty cells will be filled in though out the animation (Fig. 5.13).

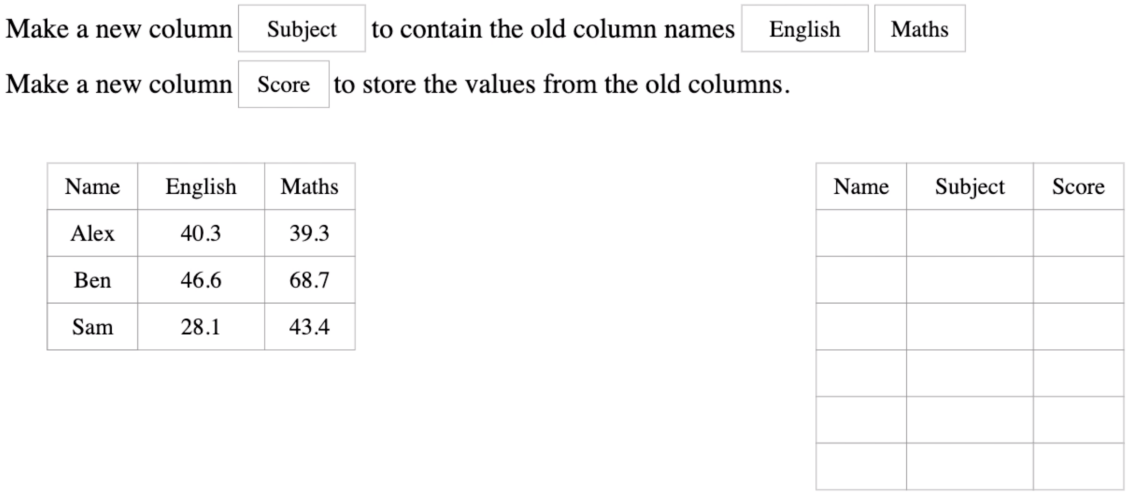


FIGURE 5.13: Wide to Long step 3

Fourth, we fade out the tables and the text in the background and show a message informing the user that the animation will now start (Fig. 5.14). For example, here we show a message "Start wide to long transformation". We fade out the background to allow the user to focus on this message.

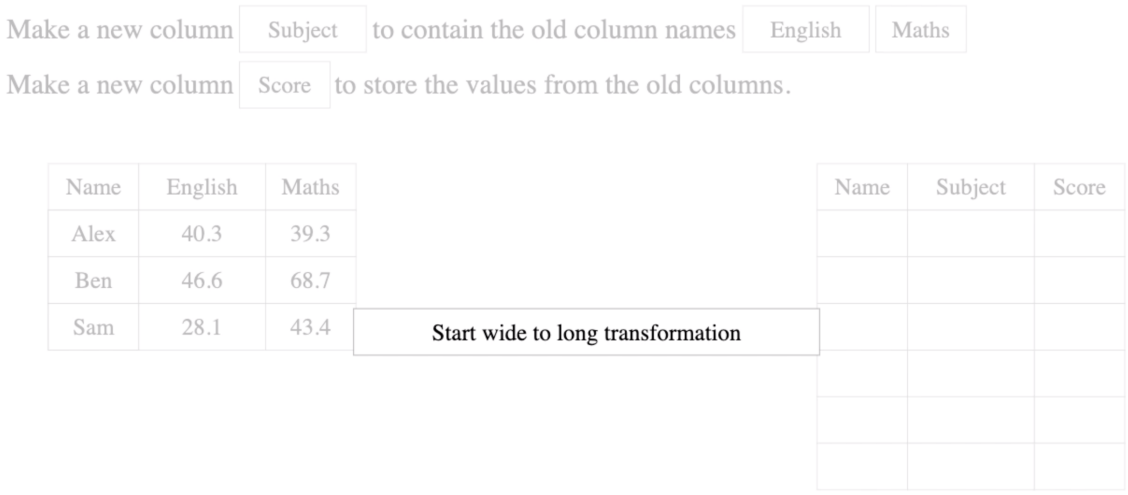


FIGURE 5.14: Wide to Long step 4

Fifth, we flash the column that is neither the key or the value column from both tables and link them with a line. We do this to catch the users attention.

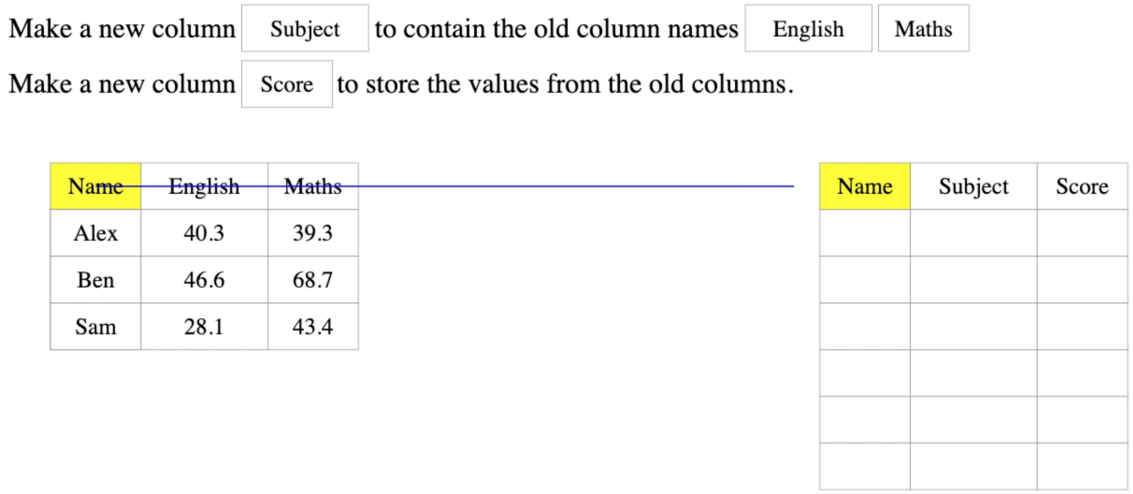


FIGURE 5.15: Wide to Long step 5

Sixth, all the values in that column will move over to the resulted table under the same column name (Fig. 5.16, caught mid-move).

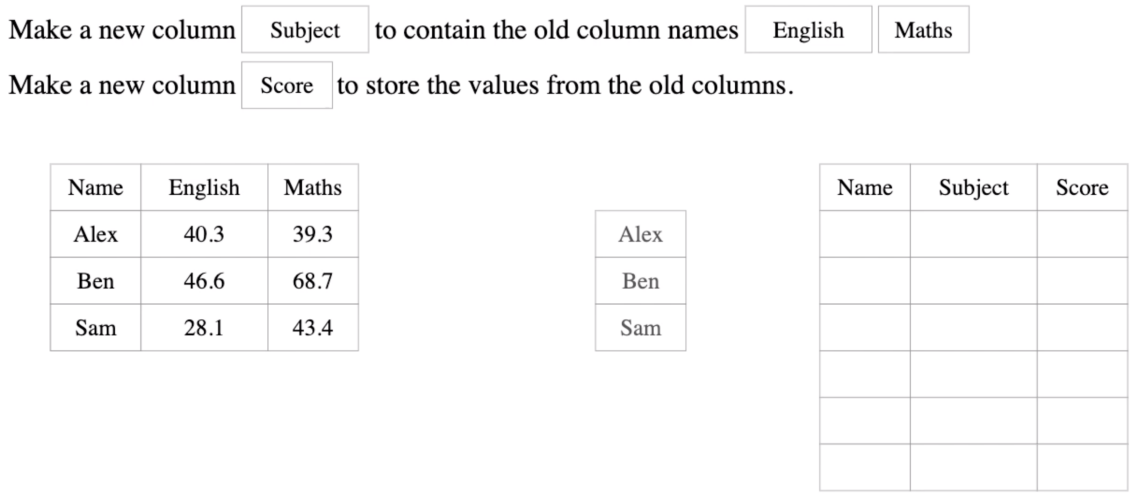


FIGURE 5.16: Wide to Long step 6

Seventh, we show a message informing the user that the column name of the first column which contains the values we wish to reshape will now move over to the result table under the column they specified (key) (Fig. 5.17).

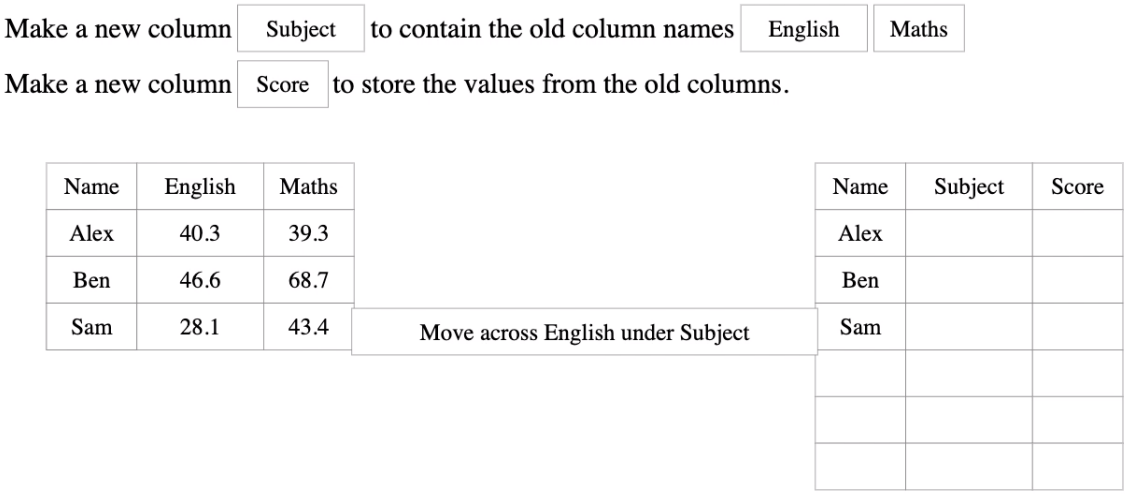


FIGURE 5.17: Wide to Long step 7

Here the we flash the column name which we are currently animating. We also flash the corresponding location of where this element should go to further emphasise this process. Lastly, we animate lines to link these flashing elements to catch the users attention (Fig. 5.18).

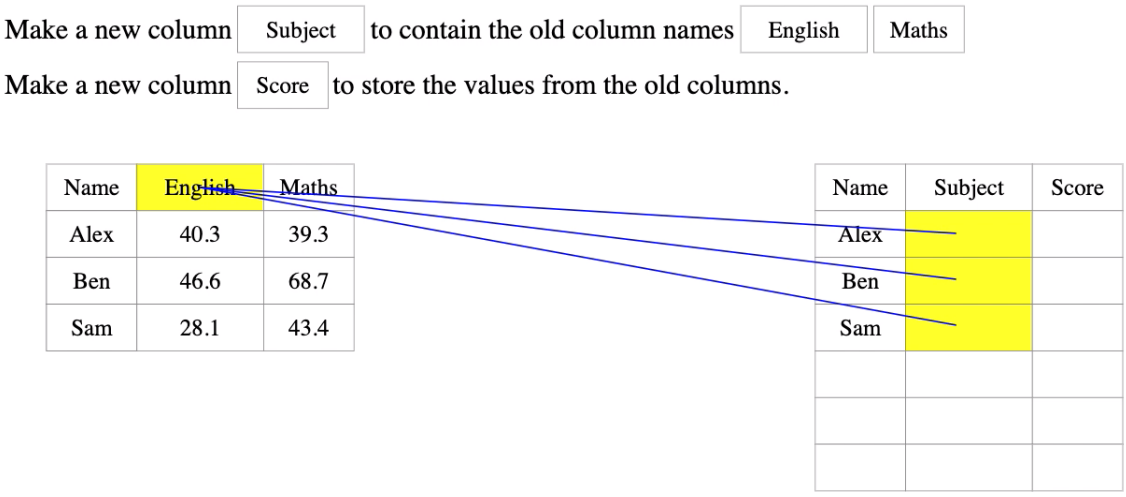


FIGURE 5.18: Wide to Long step 8

Next, we will move across the previously flashed column name from the original table to the locations that flashed in resulted table (Fig. 5.19). In this example, to compensate for the multiple locations that this element needs to go, we duplicate this element. The elements may or may not duplicate itself, this depends on the structure of the table.

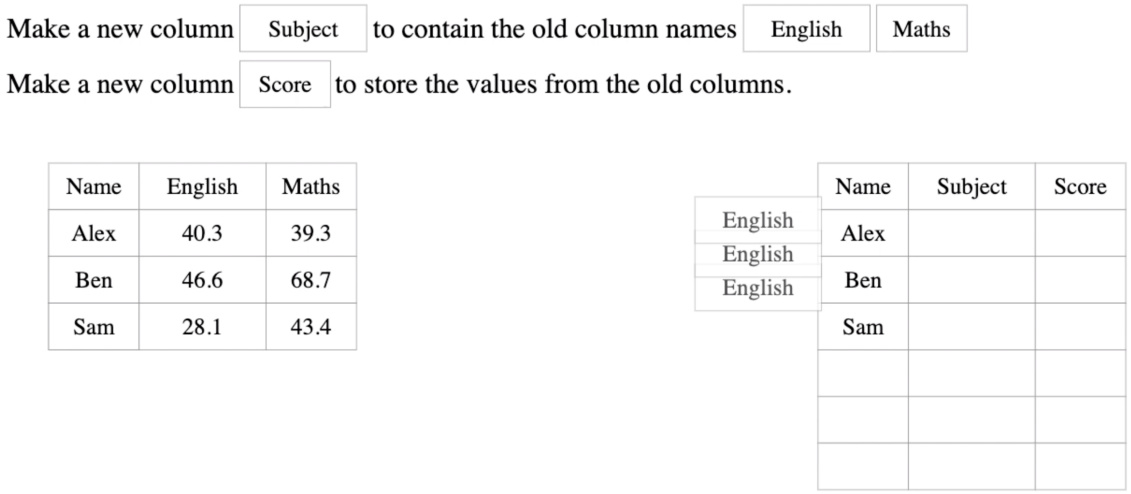


FIGURE 5.19: Wide to Long step 9

Next, we show a message to prepare the user for the next step. The next step is to move the across corresponding values that are under the previously moved column name to the resulted table. Here they are the values under in the English column (Fig. 5.20).

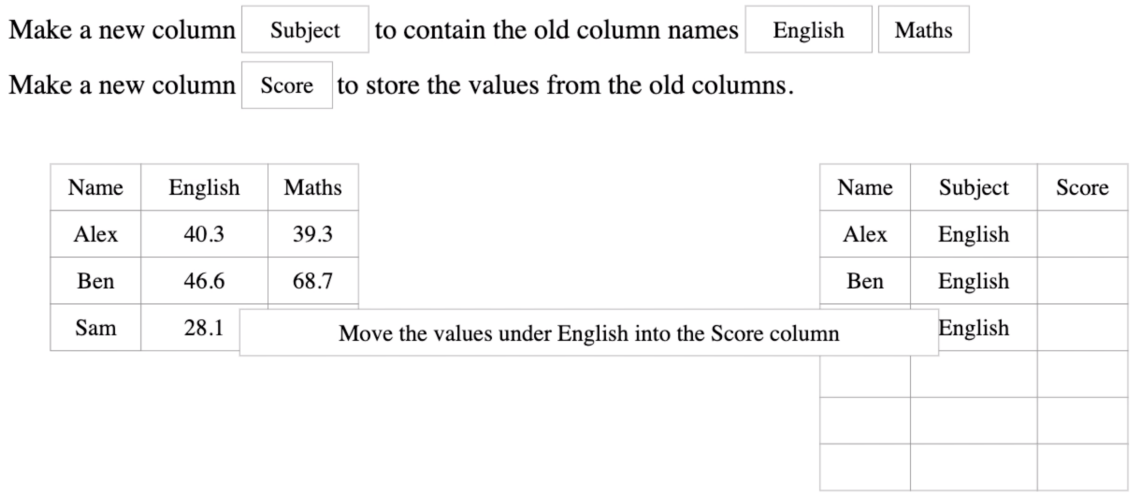


FIGURE 5.20: Wide to Long step 10

We then move these values across to the resulted table (Fig. 5.21, caught mid-move).

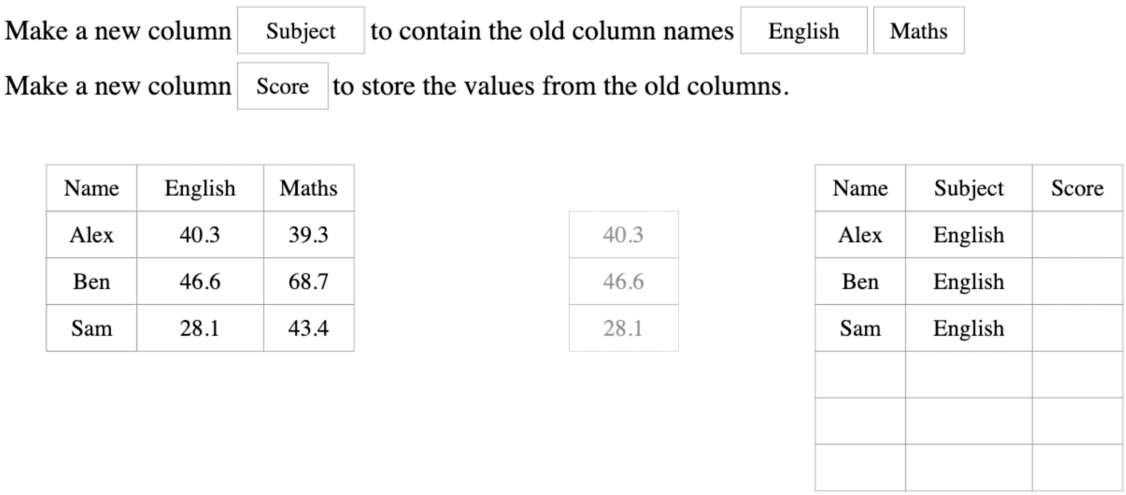


FIGURE 5.21: Wide to Long step 11

After we finish animating the first column the user wishes to reshape on. We show a message informing the user that the animation will proceed with the next column (Fig. 5.22). In this example the next column is Maths, the process for this will be the same as the previously explained steps.

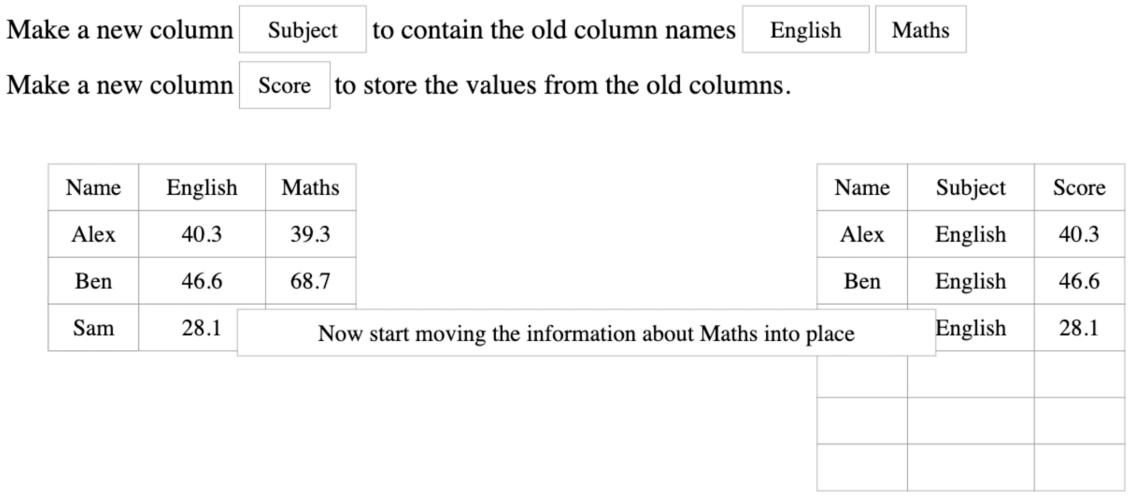


FIGURE 5.22: Wide to Long step 12

Lastly, when the animation finishes, the animation will stop (Fig. 5.23).

Make a new column Subject to contain the old column names English Maths

Make a new column Score to store the values from the old columns.

Name	English	Maths
Alex	40.3	39.3
Ben	46.6	68.7
Sam	28.1	43.4

Name	Subject	Score
Alex	English	40.3
Ben	English	46.6
Sam	English	28.1
Alex	Maths	39.3
Ben	Maths	68.7
Sam	Maths	43.4

FIGURE 5.23: Wide to Long step 13

Chapter 6

Discussion

At the beginning of the report, we have discussed the limitations of a few current approaches of teaching data joining and reshaping. The traditional method of teaching with static images is limited because they are not descriptive enough. First, they usually use dummy data sets with no meaning, or data sets with vocabulary that the learners may be unfamiliar with. Second, they assume that learners can imagine data sets or the process of the transformation in their head.

Another approach found was to teach this with animations. With this approach, the user does not have to imagine the data transformation process. However, they still use data sets which are meaningless. Although, this does solve some of the limitations that the static image approach brings, we are interested to extend this.

Therefore, we attempted develop a tool which generate animations to address all the limitations mentioned above. Additionally, this tool can also be used as a resource for the data joining and reshaping module in **iNZight** to help users understand these operations in visual way.

6.1 Future work

Though some of the animations in the software already handles complicated situations while joining or reshaping data sets, we can still expand them to handle more complicated tasks.

Additionally, we should allow students or learners to view these animations and make adjustments to the software based on their feedback.

Lastly, a SQL module is already under development. Although it is not ready at the moment but it is a good extension to our current joining module.

6.2 Conclusion

We have seen multiple approaches of teaching data joining and reshaping, they turned out to be not as efficient, mostly because of the of lack explanation in the transformation process. Therefore to overcome this problem, we used technique like highlighting, line animation, fading and instructional messages. On top of that we tried to be very careful with the ordering of these operations.

This piece of software allows users to visualise the process of data joining and reshaping. Therefore, they should be more easier to understand than most of the common approach of teaching data joining and reshaping.

6.3 Usage

The **dataAnim** is hosted on Github at https://github.com/chrk623/dataAnim. To install **dataAnim** in R:

```
devtools::install_github("chrk623\dataAnim")
```

It is suggested to view the animations in **Chrome** and have the latest version of **Rstudio**. It is also suggested to use the most up to date version of all the dependency packages.

Lastly, the shiny interactive dashboard can be found at https://chrk623.shinyapps.io/dataAnim_shiny/.

Appendix A

dataAnim Package in R

A.1 Joining Module

A.1.1 join_anim

```
#' Joining Animation
#'
#' @param join_type Type of join.
#' @param x Table x, to pass to join.
#' @param y Table y, to pass to join.
#' @param by A character value of the variable to join by.
#' @param speed Speed of the animation.
#' @param width Width of the animation frame.
#' @param height Height of the animation frame.
#' @param show_msg A logical value indicating whether to
    show instructional messages in the animation.
#' @description
#' Function to create joining animations.
#' @examples
#' data(datoy1)
#' join_anim(join_type = "left", speed = 1, x = datoy1$x,
    y = datoy1$y, by = "Name", show_msg = T)
#' myanim = join_anim(join_type = "inner", speed = 1,
    x = datoy1$x, y = datoy1$y, by = "Name", show_msg = F)
#' htmlwidgets::saveWidget(myanim)
#' @author Charco Hui
#' @import htmlwidgets
#'
#' @export
join_anim <- function(join_type = "left", x, y, by,
    speed = 1, width = NULL, height = NULL,
                      show_msg = F) {
  if(nrow(x) >= 10 || nrow(y) >= 10) {
    stop("Dataset cannot have more than 10 rows.")
  }
  if(ncol(x) >= 5 || ncol(y) >= 5) {
    stop("Dataset cannot have more than 5 columns.")
  }
  data = list(data = process_join(x = x, y = y, key = by,
    complete_action = TRUE, show_msg = show_msg,
        join_type = join_type, asJSON = TRUE),
```

```
              speed = speed, join_type = join_type)
  out = htmlwidgets::createWidget(
    name = "join_anim",
    data,
    width = width,
    height = height,
    package = 'dataAnim'
  )
  return(out)
}
#' @export
join_animOutput <- function(outputId, width = "100%",
    height = "1000") {
  shinyWidgetOutput(outputId, "join_anim", width, height,
    package = "dataAnim")
}
#' @export
join_animRender <- function(expr, env = parent.frame(),
    quoted = FALSE) {
  if (!quoted) { expr <- substitute(expr) } # force quoted
  shinyRenderWidget(expr, join_animOutput, env,
    quoted = TRUE)
}
```

A.1.2 Helper functions for joins

```
initial_prep = function(from_tbl, to_tbl, key, join_type,
    show_msg) {
  from_ind = which(colnames(from_tbl) == key)
  to_ind = which(colnames(to_tbl) == key)
  initial = sapply(to_tbl[,to_ind], FUN = function(x)
    which(from_tbl[,from_ind] == x))
  initial = lapply(initial, FUN = function(x) {
    if(length(x) == 0) {
      return(-1)
    } else {
      return(x)
    }
  })
  when = lapply(initial, getwhen) %>% Reduce(rbind, .)
    %>% as.vector()
  msg = lapply(initial, function(x) getmsg(x, join_type))
    %>% Reduce(rbind, .) %>%
    as.vector()
  msg = data.frame(name = names(initial),
    msg = as.character(msg))
  msg = msg %>% group_by(msg) %>% mutate(uq = row_number())
    %>%ungroup() %>% mutate(msg = as.character(msg)) %>%
    mutate(msg = ifelse(uq == 1, msg, NA)) %>% select(-uq)
  msg = msg %>% rowwise() %>% transmute(msg = gsub("_val_",
    name, msg)) %>% pull(msg)
```

```
  initial = tibble(row = seq(length(initial)),
    dest = initial)
  if(isTRUE(show_msg)) {
    initial = initial %>% mutate(msg = msg, when = when)
  }
  initial = initial %>% unnest() %>% split(.$row)
  return(initial)
}
find_key_col = function(obj, key, minus1 = FALSE){
  ans = which(colnames(obj) == key)
  if(minus1 == TRUE){
    return(ans - 1)
  }else{
    return(ans)
  }
}
xy_loc = function(from_tbl, to_tbl, result_tbl, key,
    minus1 = F) {
  result_tbl = result_tbl %>% mutate(stop = row_number())
  from_cn = colnames(from_tbl)
  from_tbl = from_tbl %>% mutate(start = row_number())
  out = left_join(result_tbl, from_tbl, by = from_cn) %>%
    select(start, stop) %>% na.omit() %>% arrange(stop)
  return(out)
}
xy_loc2 = function(from_tbl, to_tbl, result_tbl, key,
    minus1 = F) {
  result_tbl = result_tbl %>%
    group_by(!!!syms(colnames(to_tbl))) %>%
    mutate(addrow = row_number()) %>% ungroup()
  from_cn = colnames(from_tbl)
  from_tbl = from_tbl %>% mutate(start = row_number())
  to_cn = colnames(to_tbl)
  to_tbl = to_tbl %>% mutate(split = row_number())
  result_tbl = left_join(result_tbl, from_tbl, by = from_cn)
    %>% left_join(., to_tbl, by = to_cn) %>%
    mutate(stop = row_number())
  if(minus1 == T) {
    result_tbl = result_tbl %>%
      mutate(start = start - 1, stop = stop - 1)
  }
  out = result_tbl %>%
    select(split, start, stop, addrow) %>%
    na.omit() %>%
    split(x = ., f = .$split)
  out = lapply(out, function(xx) xx %>% select(-split))
  return(out)
}
getmsg = function(x, join_type) {
  na_msg = switch(join_type, inner = "No match for _val_.
    \nDelete", "No match for _val_")
```

```
  if(length(x) > 1) {
    return(sprintf("%s matches found for \n_val_",
        length(x)))
  }else if(x == -1) {
    return(na_msg)
  }else if(length(x) == 1) {
    return("Matching _val_")
  }
}
getwhen = function(x) {
  if(length(x) > 1) {
    return("after")
  }else if(x == -1) {
    return("after")
  }else if(length(x) == 1) {
    return("before")
  }
}
```

A.1.3 Back end code for joining

```
process_join = function(x, y, key, height = 2, width = 5,
    svg_width = 1920, svg_height = 1080,
                          complete_action = TRUE,
                              join_type = "left", show_msg,
                                  asJSON = FALSE, ...){
  xy = left_join(x, y, by = key)
  temp = list(x = include_cn(x),
              y = include_cn(y),
              xy = include_cn(xy),
              height = height,
              initial_prep = initial_prep(y ,x, key = key,
                join_type = join_type, show_msg = show_msg),
              x_key_col = find_key_col(x, key = key),
              y_key_col = find_key_col(y, key = key),
              svg_width = svg_width, svg_height = svg_height)
  if(isTRUE(complete_action)) {
    row2move = temp$initial_prep %>% Reduce(rbind, .) %>%
      filter(dest != -1) %>% pull(dest) %>%
      unique()
    row2move = outersect(1:nrow(y), row2move)
    col_ind = sapply(colnames(temp$y), function(x){
      which(colnames(temp$xy) == x)
    })
    temp = c(temp, list(com_act = list(row2move = row2move,
        col_ind = col_ind)))
  }
  temp = c(temp,
           list(x_w = col_max_width(temp$x, width = width),
                y_w = col_max_width(temp$y, width = width)))
  temp = c(temp,
```

```
          list(x_cord_x = c(0,
          cumsum(temp$x_w[-length(temp$x_w)])),
                x_cord_y = c(0,
                cumsum(temp$y_w[-length(temp$y_w)]))))
  if(asJSON == TRUE){
    return(jsonlite::toJSON(temp))
  }else{
    return(temp)
  }
}
```

A.1.4 htmlwidgets JavaScript code for `join_anim` in R

```
HTMLWidgets.widget({

  name: 'join_anim',

  type: 'output',

  factory: function (el, width, height) {
    let svg_width = width * 0.8;
    let svg_height = svg_width / 1.6;
    return {
      renderValue: function (x) {
        let data = x.data;
        let speed = x.speed;
        let join_type = x.join_type;
        let xtbl_width = arr_sum(data.x_w[0]);
        let ytbl_width = arr_sum(data.y_w[0]);
        let xy_width = xtbl_width + ytbl_width;
        let cell_height = data.height[0];
        let xscale_num = (xy_width + 1.5 * ytbl_width) / 0.9;
        let yscale_num = cell_height * (data.x.length +
            data.y.length - 1) / 0.9;
        d3.select(el)
          .append("svg")
          .attr("width", svg_width)
          .attr("height", svg_height);

        let x_scale =
          d3.scaleLinear()
          .domain([0, xscale_num])
          .range([0, svg_width]);
        let y_scale =
          d3.scaleLinear()
          .domain([0, yscale_num])
          .range([0, svg_height]);
        let height = y_scale(cell_height);

        let xtbl_start = {
          x: svg_width * 0.05,
```

```
          y: svg_height * 0.05
        };
        let ytbl_start = {
          x: svg_width - svg_width * 0.05 -
            x_scale(ytbl_width),
          y: svg_height * 0.05
        };

        let x_width = arr_scale(data.x_w[0], x_scale);
        let y_width = arr_scale(data.y_w[0], x_scale);
        let x_cord_x = arr_scale(data.x_cord_x, x_scale);
        let x_cord_y = arr_scale(data.x_cord_y, x_scale);
        let y_cord_x = Array.from(Array(data.x.length),
            (d, i) => i * height);
        let y_cord_y = Array.from(Array(data.y.length),
            (d, i) => i * height);

        x_rect_cord = draw_table(data.x, xtbl_start["x"],
            xtbl_start["y"], x_cord_x,
          x_width, height, "x", parseInt(data.x_key_col));
        y_rect_cord = draw_table(data.y, ytbl_start["x"],
            ytbl_start["y"], x_cord_y,
          y_width, height, "y", parseInt(data.y_key_col));

        let iso_action = join_type;
        if (join_type == "complete") {
          join_type = "left";
        }

        keycol_anim("x", "y", speed, start_time = 0)
          .on("end", function () {
            join_anim(data.initial_prep, speed, join_type,
                true) .on("end", () => {
                join_finalcheck(data.initial_prep)
                  .on("end", () => {
                    if (iso_action === "complete") {
                      comjoin_final(data.com_act, speed)
                        .on("end", () => {
                          iso_tbl(join_type, height, speed, 0);
                        });
                    } else {
                      iso_tbl(join_type, height, speed, 0);
                    }
                  })
              })
          })
      },
      resize: function (width, height) {}
    };
  }
});
```

A.2 Reshaping module

A.2.1 spread_anim

```
#' Spread Animation
#'
#' @param key Column used to spread out to multiple columns.
#' @param value Column containing values of the key.
#' @param data Data to pass into the function.
#' @param speed Speed of the animation.
#' @param width Width of the animation frame in pixels.
#' @param height Height of the animation frame in pixels.
#' @description
#' Long to Wide transformation.
#' @examples
#' data(toyda_long)
#' spread_anim(key = "Subject", value = "Score",
    data = toyda_long)
#' myanim = spread_anim(key = "Subject", value = "Score",
    data = toyda_long)
#' htmlwidgets::saveWidget(myanim, file = "myanim.html")
#' @author Charco Hui
#' @import htmlwidgets
#'
#' @export
spread_anim <- function(key, value, data, speed = 1,
    width = NULL, height = NULL) {
  if(ncol(data) > 3) {
    stop("Only 3 columns are supported at the moment")
  }
  data = list(data = process_spread(key = key, value = value,
    data = data, asJSON = TRUE),
                speed = speed)
  out = htmlwidgets::createWidget(
    name = "spread_anim",
    data,
    width = width,
    height = height,
    package = 'dataAnim'
  )
  return(out)
}
#' @export
spread_animOutput <- function(outputId, width = "100%",
    height = "1000") {
  shinyWidgetOutput(outputId, "spread_anim", width, height,
    package = "dataAnim")
}
#' @export
spread_animRender <- function(expr, env = parent.frame(),
    quoted = FALSE) {
```

```
  if (!quoted) { expr <- substitute(expr) }
  shinyRenderWidget(expr, spread_animOutput, env,
    quoted = TRUE)
}
```

A.2.2 gather_anim

```
#' Gather Animation
#'
#' @param key New column name to contain old columns names.
#' @param value New column name to contain old column valuess.
#' @param col Columns in the original data to transform on.
#' @param data Data to pass into the function.
#' @param speed Speed of the animation.
#' @param width Width of the animation frame in pixels.
#' @param height Height of the animation frame in pixels.
#' @description
#' Wide to long transformation.
#' @examples
#' data(toyda_wide)
#' gather_anim(key = "Subject", value = "Score",
    col = c("English", "Maths"), data = toyda_wide)
#' myanim = gather_anim(key = "Subject", value = "Score",
    col = c("English", "Maths"), data = toyda_wide)
#' htmlwidgets::saveWidget(myanim, file = "myanim.html")
#' @author Charco Hui
#' @import htmlwidgets
#'
#' @export
gather_anim <- function(key, value, data, col, speed = 1,
    width = NULL, height = NULL) {
  if(ncol(data) > 3) {
    stop("Only 3 columns are supported at the moment")
  }
  data = list(data = process_gather(key = key,
    value = value, col = col, data = data,
        asJSON = TRUE), speed = speed)
  out = htmlwidgets::createWidget(
    name = "gather_anim",
    data,
    width = width,
    height = height,
    package = 'dataAnim'
  )
  return(out)
}
#' @export
gather_animOutput <- function(outputId, width = "100%",
    height = "1000") {
  shinyWidgetOutput(outputId, "gather_anim", width,
    height, package = "dataAnim")
```

```
}
#' @export
gather_animRender <- function(expr, env = parent.frame(),
    quoted = FALSE) {
  if (!quoted) { expr <- substitute(expr) }
  shinyRenderWidget(expr, spread_animOutput, env,
    quoted = TRUE)
}
```

A.2.3 Back end code for reshaping

```
process_spread = function(key, value, data, height = 2,
    width = 5, svg_width = 1920,
                          svg_height = 1080, show_msg = T,
                            asJSON = FALSE) {
  if(ncol(data) != 3) {
    stop("Inputted dataset must have 3 columns for
        the animation to work")
  } else if(ncol(data) > 3) {
    stop("Spread animation currently only supports dataset
        with 3 columns")
  } else if(ncol(data) < 3) {
    stop("This data set is not possible to transform to long")
  }
  cn = colnames(data)
  key_ind = which(cn == key)
  value_ind = which(cn == value)
  pivot_ind = (1:ncol(data))[-c(key_ind, value_ind)]
  data = data %>% arrange(!!sym(key), !!sym(cn[pivot_ind]))
  result = data %>% spread(key = key, value = value)
  na_pos = which(is.na(result), arr.ind = TRUE)
  result[is.na(result)] = ""
  key_seq = data %>% pull(!!sym(key)) %>% as.factor()
  key_seq = cumsum(tabulate(key_seq))
  key_seq = data.frame(start = c(1,
    key_seq[-length(key_seq)] + 1), stop = key_seq)
  split_data = apply(key_seq, 1, function(x) data[x[1]:x[2],])
  key_rowseq = lapply(split_data, function(x) {
    inner_join(result %>% select(pivot_ind) %>%
        mutate(`__rn` = row_number()),
              x, by = "Name") %>% pull(`__rn`)
  })
  pivot_rows = which(!duplicated(data[,pivot_ind]))
  out = list(key_ind = key_ind, value_ind = value_ind,
    original = include_cn(data),
             height = height, svg_width = svg_width,
                pivot_ind = pivot_ind, key_seq = key_seq,
             svg_height = svg_height,
                result = include_cn(result), key = key,
                    value = value,key_rowseq = key_rowseq,
                        pivot_rows = pivot_rows,
```

```
                pivot = cn[pivot_ind], na_pos = na_pos)
  out = c(out, list(rslt_cn = as.vector(colnames(out$result)),
                    og_w = col_max_width(out$original,
                        width = width),
                    rslt_w = col_max_width(out$result,
                        width = width)))
  out = c(out, list(og_cord_x = c(0,
    cumsum(out$og_w[-length(out$og_w)])),rslt_cord_x =
        c(0, cumsum(out$rslt_w[-length(out$rslt_w)]))))
  if(isTRUE(asJSON)) {
    return(jsonlite::toJSON(out))
  } else {
    return(out)
  }
}
process_gather = function(key, value, col, data, height = 2,
    width = 5, svg_width = 1920,
                          svg_height = 1080, show_msg = T,
                            asJSON = FALSE) {
  if(missing(col)) {
    stop("Columns to keep must be specified.")
  }
  cn = colnames(data)
  pivot_ind = which(!(cn %in% col))
  col_ind = which(cn %in% col)
  result = data %>% gather(key = !!sym(key),
    value = !!sym(value), col)
  cn2 = colnames(result)
  key_ind = which(cn2 == key)
  value_ind = which(cn2 == value)
  out = list(original = include_cn(data), height = height,
    svg_height = svg_height, svg_width = svg_width,
        result = include_cn(result), pivot_ind = pivot_ind,
             col_ind = col_ind, key = key, value = value,
                col = col, value_ind = value_ind,
             key_ind = key_ind)
  out = c(out, list(rslt_cn = as.vector(colnames(out$result)),
                    og_w = col_max_width(out$original,
                        width = width),
                    rslt_w = col_max_width(out$result,
                        width = width)))
  out = c(out, list(og_cord_x = c(0,
    cumsum(out$og_w[-length(out$og_w)])),
                    rslt_cord_x = c(0,
                        cumsum(out$rslt_w
                            [-length(out$rslt_w)]))))
  if(isTRUE(asJSON)) {
    return(jsonlite::toJSON(out))
  } else {
    return(out)
  }
```

```
}
```

A.2.4 htmlwidgets JavaScript code for `spread_anim` R

```
HTMLWidgets.widget({
    name: 'spread_anim',
    type: 'output',
    factory: function (el, width, height) {
      let svg_width = width * 0.8;
      let svg_height = svg_width / 1.6;
      return {
        renderValue: function (x) {
          let data = x.data;
          svg_div = "animpanel0";
          let otbl_width = arr_sum(data.og_w[0]);
          let rtbl_width = arr_sum(data.rslt_w[0]);
          let or_width = otbl_width + rtbl_width;
          let cell_height = data.height[0];
          let xscale_num = (or_width + 1.5 * rtbl_width) / 0.9;
          let yscale_num = cell_height * (data.original.length +
            data.original.length - 1) / 0.9;
          el = d3.select("body")
            .append("div")
            .attr("id", svg_div);
          el.append("svg")
            .attr("width", svg_width)
            .attr("height", svg_height);
          let x_scale =
            d3.scaleLinear()
            .domain([0, xscale_num])
            .range([0, svg_width]);
          let y_scale =
            d3.scaleLinear()
            .domain([0, yscale_num])
            .range([0, svg_height]);
          let height = y_scale(cell_height);
          let otbl_start = {
            x: svg_width * 0.05,
            y: svg_height * 0.25
          };
          let ptxt_start = {
            x: svg_width * 0.015,
            y: svg_height * 0.05
          };
          let o_width = arr_scale(data.og_w[0], x_scale);
          let x_cord_o = arr_scale(data.og_cord_x,
            x_scale);
          let rslt_cord_x = arr_scale(data.rslt_cord_x, x_scale);
          let r_width = arr_scale(data.rslt_w[0], x_scale);
          let r_tbl_tl = {
            x: svg_width - svg_width * 0.05 - arr_sum(r_width),
```

```
          y: otbl_start["y"]
        };
        let key_rect_w = x_scale(data.rslt_w[0]
          [data.key_ind[0] - 1]);
        let value_rect_w = x_scale(data.rslt_w[0]
          [data.value_ind[0] - 1]);
        let tbl_mid_xy = {
          x: svg_width / 2,
          y: svg_height / 2
        };
        draw_table_rectonly(data.result, r_tbl_tl["x"],
          r_tbl_tl["y"], rslt_cord_x,
          r_width, height, "r", parseInt(data.pivot_ind),
              false);
        d3.selectAll(".r_rows")
          .style("opacity", 0);
        input_text_node = d3.select(".r_rows").
          selectAll(".r_cols").nodes();
        input_text_node.forEach(function (d, i) {
          cur_rect = d3.select(input_text_node[i])
            .select("rect");
          d3.select(input_text_node[i])
            .append("text")
            .attr("x", parseFloat(cur_rect.attr("x")) +
              parseFloat(cur_rect.
                  attr("width") / 2))
            .attr("y", parseFloat(cur_rect.attr("y")) +
              parseFloat(cur_rect.
                  attr("height") / 2))
            .style("font-size", height * 0.5)
            .text(data.rslt_cn[i]);
        })
        x_rect_cord = draw_table(data.original,
          otbl_start["x"], otbl_start["y"], x_cord_o,
          o_width, height, "o", parseInt(data.pivot_ind));
        let all_o_rows = d3.selectAll(".o_rows")
          .nodes();
        let delay_time = 2000;
        let msg = true;
        let speed = data.speed;
        let new_col_cnt = 2;
        prepare_spread(ptxt_start,
          data.key[0], data.value[0], data.key_ind[0],
              key_rect_ w   value_rect_w, data.key_seq,
                  height, delay_time, speed)
                      .on("end", function () {
            d3.selectAll(".r_rows").style("opacity", 1)
            spread_anim_pivot(tbl_mid_xy, data.pivot[0],
              data.pivot_rows, speed, delay_time, msg = true)
              .on("end", function () {
                d3.selectAll(".removed").remove();
```

```
                data.key_seq.forEach(function (d, i) {
                  current_chunk = all_o_rows.
                      slice(d["start"], d["stop"] + 1);
                  delay_time = spread_anim_move(tbl_mid_xy,
                      current_chunk, data.key_rowseq[i],
                    data.key_ind[0], new_col_cnt,
                      data.value_ind[0], speed, delay_time,
                          msg);
                  msg = false;
                  new_col_cnt++;
                })
                spread_anim_fillna(data.na_pos, speed,
                  delay_time);
              })
          });
      },
      resize: function (width, height) {}
    };
  }
});
```

A.2.5 htmlwidgets JavaScript code for `gather_anim` R

```
HTMLWidgets.widget({
    name: 'gather_anim',
    type: 'output',
    factory: function (el, width, height) {
        let svg_width = width * 0.8;
        let svg_height = svg_width / 1.6;
        return {
            renderValue: function (x) {
                let data = x.data;
                svg_div = "animpanel0";
                let otbl_width = arr_sum(data.og_w[0]);
                let rtbl_width = arr_sum(data.rslt_w[0]);
                let or_width = otbl_width + rtbl_width;
                let cell_height = data.height[0];
                let xscale_num = (or_width + 1.5 *
                    rtbl_width) / 0.9;
                let yscale_num = cell_height *
                    (data.result.length +  data.result.length
                        - 1) / 0.9;
                el = d3.select("body")
                    .append("div")
                    .attr("id", svg_div);
                el.append("svg")
                    .attr("width", svg_width)
                    .attr("height", svg_height);
                let x_scale =
                    d3.scaleLinear()
                    .domain([0, xscale_num])
```

```
        .range([0, svg_width]);
    let y_scale =
        d3.scaleLinear()
        .domain([0, yscale_num])
        .range([0, svg_height]);
    let height = y_scale(cell_height);
    let otbl_start = {
        x: svg_width * 0.05,
        y: svg_height * 0.25
    }
    let ptxt_start = {
        x: svg_width * 0.015,
        y: svg_height * 0.05
    }
    let o_width = arr_scale(data.og_w[0],
        x_scale);
    let x_cord_o = arr_scale(data.og_cord_x,
        x_scale);
    let rslt_cord_x = arr_scale(data.rslt_cord_x,
        x_scale);
    let r_width = arr_scale(data.rslt_w[0],
        x_scale);
    let r_tbl_tl = {
        x: svg_width - svg_width * 0.05 -
            arr_sum(r_width),
        y: otbl_start["y"]
    };
    let key_rect_w = x_scale(data.rslt_w[0]
        [data.key_ind[0] - 1]);
    let value_rect_w = x_scale(data.rslt_w[0]
        [data.value_ind[0] - 1]);
    let tbl_mid_xy = {
        x: svg_width / 2,
        y: svg_height / 2
    };
    draw_table_rectonly(data.result,
        r_tbl_tl["x"], r_tbl_tl["y"], rslt_cord_x,
        r_width, height, "r",
            parseInt(data.pivot_ind));
    d3.selectAll(".r_rows")
        .style("opacity", 0);
    input_text_node = d3.select(".r_rows").
        selectAll(".r_cols").nodes();
    input_text_node.forEach(function (d, i) {
        cur_rect = d3.select(input_text_node[i])
            .select("rect");
        d3.select(input_text_node[i])
            .append("text")
            .attr("x", parseFloat
                (cur_rect.attr("x")) +
                    parseFloat(cur_rect.
```

```
                                attr("width") / 2)).attr("y",
                        parseFloat(cur_rect.attr("y")) +
                            parseFloat(cur_rect.
                                attr("height") / 2))
                    .style("font-size", height * 0.5)
                    .text(data.rslt_cn[i]);
            })
            x_rect_cord = draw_table(data.original,
                otbl_start["x"], otbl_start["y"], x_cord_o,
                o_width, height, "o",
                    parseInt(data.pivot_ind));
            let all_o_rows = d3.selectAll(".o_rows")
                .nodes();
            let all_r_rows = d3.selectAll(".r_rows")
                .nodes();
            let speed = 1;
            let chunk_increment = all_o_rows.length - 1;
            let start_chunk = 1;
            let delay_time = 0;
            let tran_time = 2500 / speed;
            let msg = true;
            let first = true;
        prepare_gather(ptxt_start, data.col_ind,
           data.key[0], data.value[0], key_rect_w,
                value_rect_w, height, delay_time, 1)
           .on("end", function () {
                data.col_ind.forEach(function (d, i) {
                    end_chunk = start_chunk +
                    chunk_increment;
                    current_chunk = all_r_rows.
                    slice(start_chunk, end_chunk)
                    delay_time = gather_anim_pivot
                    (tbl_mid_xy, current_chunk, 1, 1,
                    speed, delay_time, msg, first);
                    delay_time = gather_anim_key
                        (tbl_mid_xy, current_chunk,
                            data.col[i], d, data.key[0],
                                data.key_ind[0], speed,
                                    delay_time, msg);
                    delay_time = gather_anim_value
                        (tbl_mid_xy, current_chunk, data.col[i],d,
                            data.value[0], data.value_ind[0],
                                speed, delay_time, msg);
                    if (data.col.length - 1 != i) {
                        iso_msgbox(tbl_mid_xy["x"],
                            tbl_mid_xy["y"], height * 15,
                                height,`Now start moving
                                    the information about
                                        ${data.col[i + 1]} into
                                    place `, delay_time,
                                        tran_time,  0.5,
```

```
                                                tran_time / 2);
                    delay_time = delay_time + tran_time;
                            }
                            msg = false;
                            first = false;
                            start_chunk = end_chunk;
                        })
                    });
            },
            resize: function (width, height) {}
        };
    }
});
```

A.3 Helper function for both modules

```
include_cn = function(obj, ...){
  obj = as_tibble(apply(obj, 2, function(x) {
    if(is.numeric(x)) {
      x = round(x, 3)
    }
    return(x)
  }))
  return(rbind(colnames(obj), obj))
}
outersect = function(x, y){
  sort(c(setdiff(x, y),
         setdiff(y, x)))
}
col_max_width = function(obj, width, ...){
  obj = as_tibble(apply(obj, 2, as.character))
  obj = obj %>% transmute_all(nchar) %>%
    summarise_all(max, na.rm = T) %>% as.matrix()
  obj = ifelse(obj < width, width, obj)
  return(obj)
}
tbl_cord_x = function(x) {
  x = cumsum(x)
  return(c(0, x[-length(x)]))
}
is.integer0 <- function(x){
  is.integer(x) && length(x) == 0L
}
prepare_table = function(data, height = 3, width = 3) {
  data = include_cn(data)
  col_width = col_max_width(data)
  col_width = ifelse(col_width < width, width, col_width)
  return(list(data = data, col_width = col_width,
              height = height,
              cord_x = tbl_cord_x(col_width)))
}
```

```
num_colcharacter = function(df, key) {
  if(missing(key)) {
    stop("Must supply key")
  }
  df = df %>% select(-key)
  info = sapply(df, class)
  return(max(sum(info == "numeric"), sum(info == "factor")))
  }
```

Appendix B

JavaScript code for dataAnim

B.1 Joining module

```
function clone_nodes() {
    d3.select(this).clone(false);
};

function get_translation(obj, type) {
    if (type == "node") {
        string = d3.select(obj).attr("transform");
    } else {
        string = obj.attr("transform");
    }
    if (string === null) {
        return ([0, 0]);
    } else {
        string = string.substring(string.indexOf("(") + 1,
            string.indexOf(")")).split(",");
        return string.map((d, i) => parseInt(d));
    }
}

function clone_everything(selector) {
    var node = d3.select(selector).node();
    return d3.select(node.parentNode.insertBefore(node
        .cloneNode(true), node.nextSibling));
};

function a_intersect(a, b) {
    var t;
    if (b.length > a.length) t = b, b = a, a = t;
    return a.filter(function (e) {
        return b.indexOf(e) > -1;
    });
}

function a_intersect2(a, b) {
    var t;
    if (b.length > a.length) t = b, b = a, a = t;
    return a.filter(function (e) {
```

```
        return b.indexOf(e) > -1;
    }).filter(function (e, i, c) {
        return c.indexOf(e) === i;
    });
}

function a_outersect(a, b) {
    return a.filter(x => !b.includes(x))
}

function arr_last(arr) {
    return arr.slice(-1)[0];
}

function arr_nth(arr, n) {
    return arr.filter(function (value, index, ar) {
        index++;
        return (index % n == 0);
    });
}

function arr_sum(arr) {
    let reducer = (accumulator, currentValue) =>
        accumulator + currentValue;
    return arr.reduce(reducer);
}

function arr_scale(arr, sc) {
    arr = arr.map((d, i) => {
        return (sc(d))
    });
    return arr;
}

function parent_setclass(sel, s_class, level) {
    for (var i = 0; i < level; i++) {
        sel = sel.select(function () {
            return this.parentNode;
        });
        sel.classed(s_class, true);
    }
}

function db_pulse_rect(nodes, col, tran_time = 1000,
    delay_time = 0, keep_highlighted = false) {
    if (nodes.length === undefined) {
        let og_col = d3.select(nodes).style("fill");
        for (var j = 0; j < 4; j++) {
            if (j % 2 === 0) {
                d3.select(nodes)
                    .transition()
```

```
                    .duration(tran_time / 4)
                    .delay(delay_time + tran_time / 4 * j)
                    .style("fill", col);
            } else {
                d3.select(nodes)
                    .transition()
                    .duration(tran_time / 4)
                    .delay(delay_time + tran_time / 4 * j)
                    .style("fill", og_col);
            }
        }
        if (keep_highlighted === true) {
            d3.select(nodes)
                .transition()
                .delay(delay_time + tran_time)
                .style("fill", col);
        }
    } else {
        nodes.forEach(function (d, i) {
            for (var j = 0; j < 4; j++) {
                if (j % 2 === 0) {
                    d3.select(d)
                        .transition()
                        .duration(tran_time / 4)
                        .delay(delay_time + tran_time /
                            4 * j)
                        .style("fill", col);
                } else {
                    d3.select(d)
                        .transition()
                        .duration(tran_time / 4)
                        .delay(delay_time + tran_time /
                            4 * j)
                        .style("fill", null);
                }
            }
            if (keep_highlighted === true) {
                d3.select(d)
                    .transition()
                    .delay(delay_time + tran_time)
                    .style("fill", col);
            }
        })
    }
}

function rect_mid_cord(nodes) {
    let result = new Array();
    if (nodes.length == undefined) {
        x = parseFloat(d3.select(nodes).attr("x"));
        y = parseFloat(d3.select(nodes).attr("y"));
```

```
        wd = parseFloat(d3.select(nodes).attr("width"));
        ht = parseFloat(d3.select(nodes).attr("height"));
        result.push({
            x: x + wd / 2,
            y: y + ht / 2
        });
    } else {
        nodes.forEach(function (d, i) {
            x = parseFloat(d3.select(d).attr("x"));
            y = parseFloat(d3.select(d).attr("y"));
            wd = parseFloat(d3.select(d).attr("width"));
            ht = parseFloat(d3.select(d).attr("height"));
            result.push({
                x: x + wd / 2,
                y: y + ht / 2
            });
        })
    }
    return result;
}

function nkeycol_width(id) {
    let temp_nodes = d3.select(`.${id}_rows`)
        .selectAll(`.${id}_cols:not(.${id}_cols_key`)
        .select("rect").nodes();
    let store = new Array()
    temp_nodes.forEach(function (d, i) {
        store.push(parseFloat(d3.select(d).attr("width")));
    })
    return store;
}

function link_rect_line(base, base_key_rect_xy, dest = -1,
    qm_align = "left", tran_time = 2000, delay_time = 0,
    msg, removeall = true, msg_pause = 2000) {
    let rect_height = parseFloat(d3.select("rect")
        .attr("height"));
    let rect_width = null;
    d3.select(".x_rows")
        .selectAll("rect")
        .each(function (d, i) {
            cur_width = d3.select(this).attr("width");
            if (i == 0) {
                rect_width = cur_width;
            } else {
                if (cur_width < rect_width) {
                    rect_width = cur_width;
                }
            }
        });
    let cond = dest;
```

```
    let qm_tran_time = tran_time * 0.8;
    let qm_delay_time = qm_tran_time;
    let base_node = base;
    let dest_node = dest;
    let mid_cord = tbl_mid_cord("x", "y");
    let msg_trantime = tran_time / 2;
    db_pulse_rect(base_node, "yellow", tran_time, delay_time
        , true);
    if (cond != -1) {
        db_pulse_rect(dest_node, "yellow", tran_time,
            delay_time, true);
    }
    base = rect_mid_cord(base)[0];
    if (cond == -1) {
        dest = [tbl_mid_cord("x", "y")]
    } else {
        dest = rect_mid_cord(dest);
    }
    if (qm_align == "right") {
        qm_class = "qm";
    } else if (qm_align === "left") {
        qm_class = "qm2";
    }
    let link_line = new Array();
    dest.forEach(function (d, i) {
        link_line_i = d3.select("svg")
            .append("line")
            .attr("x1", base_key_rect_xy["x"])
            .attr("y1", base_key_rect_xy["y"])
            .attr("x2", base_key_rect_xy["x"])
            .attr("y2", base_key_rect_xy["y"]);
        link_line.push(link_line_i);
    })
    if (cond != -1) {
        link_line.forEach(function (d, i) {
            d.transition()
                .delay(delay_time)
                .duration(tran_time)
                .attr("x2", dest[i]["x"])
                .attr("y2", dest[i]["y"]);
        })
        delay_time = delay_time + tran_time;
    } else {
        link_line.forEach(function (d, i) {
            d.transition()
                .delay(delay_time)
                .duration(tran_time)
                .attr("x2", dest[i]["x"])
                .attr("y2", dest[i]["y"]);
        })
        delay_time = delay_time + tran_time;
```

```
        qm = d3.select("svg")
            .append("text")
            .attr("class", qm_class)
            .attr("x", dest[0]["x"])
            .attr("y", dest[0]["y"])
            .style("font-size", 0);
        qm.transition()
            .delay(delay_time)
            .duration(qm_tran_time)
            .text("?")
            .style("font-size", rect_height * 1.5)
        delay_time = delay_time + qm_tran_time;
    }
    if (msg["msg"] != undefined) {
        let msg_x = new Array();
        let msg_y = new Array();
        link_line.forEach(function (d, i) {
            msg_x.push((dest[0]["x"] + base_key_rect_xy["x"])
                / 2);
            msg_y.push((dest[0]["y"] + base_key_rect_xy["y"])
                / 2);
        });
        msg_x = arr_sum(msg_x) / msg_x.length;
        msg_y = arr_sum(msg_y) / msg_y.length;
        delay_time = msg_box(msg_x, msg_y, rect_width * 3
            ,rect_height * 2, msg["msg"], delay_time,
            msg_trantime, msg_pause);
        delay_time = delay_time + tran_time / 2;
    }
    link_line.forEach(function (d, i) {
        d.transition()
            .delay(delay_time)
            .on("end", function () {
                d3.select(this)
                    .remove();
                d3.selectAll("rect")
                    .style("fill", null);
            });
    })
    if (cond == -1) {
        qm.transition()
            .delay(delay_time)
            .on("end", function () {
                d3.select(this)
                    .remove();
            });
    }
    return delay_time;
}

function link_rect_line2(base, base_key_rect_xy, dest = -1
```

```
    ,qm_align = "left", tran_time = 2000, delay_time = 0,
    msg, removeall = true) {
    if (msg["msg"] != undefined) {
        msg_trantime = tran_time * 0.15;
        msg_pausetime = tran_time * 0.35;
        tran_time = tran_time / 2;
    }
    if (dest == -1) {
        qm_tran_time = tran_time * 0.2;
        tran_time = tran_time * 0.8;
    }
    let rect_height = parseFloat(d3.select("rect")
        .attr("height"));
    let rect_width = null;
    d3.select(".x_rows")
        .selectAll("rect")
        .each(function (d, i) {
            cur_width = d3.select(this).attr("width");
            if (i == 0) {
                rect_width = cur_width;
            } else {
                if (cur_width < rect_width) {
                    rect_width = cur_width;
                }
            }
        });
    let cond = dest;
    let base_node = base;
    let dest_node = dest;
    let mid_cord = tbl_mid_cord("x", "y");
    db_pulse_rect(base_node, "yellow", tran_time, delay_time
        ,true);
    if (cond != -1) {
        db_pulse_rect(dest_node, "yellow", tran_time,
            delay_time, true);
    }
    base = rect_mid_cord(base)[0];
    if (cond == -1) {
        dest = [tbl_mid_cord("x", "y")]
    } else {
        dest = rect_mid_cord(dest);
    }
    if (qm_align == "right") {
        qm_class = "qm";
    } else if (qm_align === "left") {
        qm_class = "qm2";
    }
    let link_line = new Array();
    dest.forEach(function (d, i) {
        link_line_i = d3.select("svg")
            .append("line")
```

```
            .attr("x1", base_key_rect_xy["x"])
            .attr("y1", base_key_rect_xy["y"])
            .attr("x2", base_key_rect_xy["x"])
            .attr("y2", base_key_rect_xy["y"]);
        link_line.push(link_line_i);
    })
    if (cond != -1) {
        link_line.forEach(function (d, i) {
            d.transition()
                .delay(delay_time)
                .duration(tran_time)
                .attr("x2", dest[i]["x"])
                .attr("y2", dest[i]["y"]);
        })
        delay_time = delay_time + tran_time;
    } else {
        link_line.forEach(function (d, i) {
            d.transition()
                .delay(delay_time)
                .duration(tran_time)
                .attr("x2", dest[i]["x"])
                .attr("y2", dest[i]["y"]);
        })
        delay_time = delay_time + tran_time;
        qm = d3.select("svg")
            .append("text")
            .attr("class", qm_class)
            .attr("x", dest[0]["x"])
            .attr("y", dest[0]["y"])
            .style("font-size", 0);
        qm.transition()
            .delay(delay_time)
            .duration(qm_tran_time)
            .text("?")
            .style("font-size", rect_height * 1.5)
        delay_time = delay_time + qm_tran_time;
    }
    if (msg["msg"] != undefined) {
        let msg_x = new Array();
        let msg_y = new Array();
        link_line.forEach(function (d, i) {
            msg_x.push((dest[0]["x"] + base_key_rect_xy["x"])
                / 2);
            msg_y.push((dest[0]["y"] + base_key_rect_xy["y"])
                / 2);
        });
        msg_x = arr_sum(msg_x) / msg_x.length;
        msg_y = arr_sum(msg_y) / msg_y.length;
        delay_time = msg_box(msg_x, msg_y, rect_width * 3
            ,rect_height * 2, msg["msg"], delay_time,
            msg_trantime, msg_pausetime);
```

```
    }
    link_line.forEach(function (d, i) {
        d.transition()
            .delay(delay_time)
            .on("end", function () {
                d3.select(this)
                    .remove();
                d3.selectAll("rect")
                    .style("fill", null);
            });
    })
    if (cond == -1) {
        qm.transition()
            .delay(delay_time)
            .on("end", function () {
                d3.select(this)
                    .remove();
            });
    }
    return delay_time;
}

function link_rectline_only(base, base_key_rect_xy
    ,dest = -1, tran_time = 2000, delay_time = 0) {
    let rect_height = parseFloat(d3.select("rect")
        .attr("height"));
    let cond = dest;
    let qm_tran_time = tran_time * 0.8;
    let return_delay = delay_time + tran_time;
    let base_node = base;
    let dest_node = dest;
    base = rect_mid_cord(base)[0];
    if (cond == -1) {
        dest = [tbl_mid_cord("x", "y")]
        return_delay = return_delay + qm_tran_time;
    } else {
        dest = rect_mid_cord(dest)
    }
    dest.forEach(function (d, i) {
        link_line = d3.select("svg")
            .append("line");
        link_line.attr("x1", base_key_rect_xy["x"])
            .attr("y1", base_key_rect_xy["y"])
            .attr("x2", base_key_rect_xy["x"])
            .attr("y2", base_key_rect_xy["y"]);
        link_line
            .transition()
            .duration(tran_time)
            .delay(delay_time)
            .attr("x2", d["x"])
            .attr("y2", d["y"])
```

```
            .on("end", function () {
                link_line = d3.select(this);
                d3.select("svg")
                    .transition()
                    .delay(1000)
                    .on("end", function ()
                        {
                        link_line.remove()
                    })
            });
    })
    return return_delay;
}

function move_rect(to_sel, from_node, tran_time
    ,delay_time) {
    parent_setclass(d3.select(from_node), "moved", 1);
    d3.select(from_node)
        .select("rect")
        .classed("moved", true)
        .transition()
        .delay(delay_time)
        .duration(tran_time)
        .on("start", function (d, i) {
            d3.select(this)
                .clone(true);
        })
        .attr("x", to_sel.attr("x"))
        .attr("y", to_sel.attr("y"))
        .attr("width", to_sel.attr("width"))
        .on("end", function (d, i) {
            d3.select(this)
                .remove();
        });
}

function move_text(to_sel, from_node, tran_time, delay_time) {
    d3.select(from_node)
        .select("text")
        .classed("moved", true)
        .transition()
        .delay(delay_time)
        .duration(tran_time)
        .on("start", function (d, i) {
            d3.select(this)
                .clone(true);
        })
        .attr("x", to_sel.attr("x"))
        .attr("y", to_sel.attr("y"))
        .on("end", function (d, i) {
            d3.select(this)
```

```
                .remove();
        });
}

function movexy_rect(node, x, y, width
    ,tran_time, delay_time,
    insert_txt = null, remove = false) {
    if (y === null) {
        y = d3.select(node)
            .select("rect")
            .attr("y")
    }
    if (insert_txt != null || insert_txt != undefined) {}
    node = d3.select(node)
        .select("rect")
        .clone(true)
        .node();
    d3.select(".x_rows")
        .insert(function () {
            return node;
        })
    d3.select(node).classed("moved", true)
        .transition()
        .delay(delay_time)
        .duration(tran_time)
        .attr("x", x)
        .attr("y", y)
        .attr("width", width)
        .on("end", function (d, i) {
            if (remove === true) {
                d3.select(this)
                    .remove();
            }
        });
}

function movexy_text(node, x, y, adj_y = null, width,
        tran_time, delay_time,insert_txt = null,
            remove = false) {
    parent_setclass(d3.select(node), "moved", 1);
    if (y === null) {
        y = d3.select(node)
            .select("text")
            .attr("y");
    }
    if (adj_y !== null) {
        y = y + 0.5 * adj_y;
    }
    node = d3.select(node)
        .select("text")
        .clone(true);
```

```
    node.classed("moved", true)
        .transition()
        .delay(delay_time)
        .duration(tran_time)
        .attr("x", parseInt(x) + 0.5 * width)
        .attr("y", y)
        .on("end", function (d, i) {
            if (remove === true) {
                d3.select(this)
                    .remove();
            }
        });
}

function movexy_cell(node, rx, ry, tx, ty, adj_ty = null,
    width ,tran_time, delay_time, location = null,
    msg = undefined, remove = false) {
    rnode = d3.select(node)
        .select("rect")
        .clone(true)
        .node();
    tnode = d3.select(node)
        .select("text")
        .clone(true)
        .node();
    if (ry === null) {
        ry = d3.select(node)
            .select("rect")
            .attr("y")
    }
    d3.select(rnode)
        .transition()
        .delay(delay_time)
        .duration(tran_time)
        .attr("x", rx)
        .attr("y", ry)
        .attr("width", width)
        .on("end", function (d, i) {
            if (location != null) {
                base = this;
                d3.select(location)
                    .insert(function () {
                        return base;
                    })
            }
        });
    if (ty === null) {
        ty = d3.select(node)
            .select("text")
            .attr("y");
    }
```

```
    if (adj_ty !== null) {
        ty = ty + 0.5 * adj_ty;
    }
    d3.select(tnode)
        .transition()
        .delay(delay_time)
        .duration(tran_time)
        .attr("x", parseFloat(tx) + 0.5 * width)
        .attr("y", ty)
        .on("end", function (d, i) {
            if (location != null) {
                base = this;
                d3.select(location)
                    .insert(function () {
                        return base;
                    })
            }
        });
    return delay_time + tran_time;
}

function movexy_cell_wobj(node, xy, height, width
    ,tran_time, delay_time,
    location = null, join_type = "left", msg = undefined
        ,remove = false) {
    let xy_keys = Object.keys(xy);
    let return_delay = 0;
    if (node.length == undefined) {
        node = [node];
    }
    if (node[0] == undefined) {
        if (join_type === "left" || join_type === "complete") {
            xy[0]["x"].forEach(function (d, i) {
                na_rects(d, xy[0]["y"], width[i], height,
                `.x_rows:nth-child(${location[0]})`,
                tran_time, delay_time)
            });
        } else if (join_type === "inner") {
            d3.select(`.x_rows:nth-child(${location[0]})`)
                .classed("remove_row", true)
                .transition()
                .delay(delay_time)
                .duration(tran_time)
                .style("opacity", 0);
        }
        return_delay = return_delay + tran_time;
        return return_delay;
    }
    node.forEach(function (d, i) {
        cur_node = d3.select(node[i]).selectAll(".y_cols:not
            (.y_cols_key)").nodes();
```

```
        cur_node.forEach(function (d2, j) {
            movexy_cell(d3.select(d2).node(), xy[i]["x"][j]
                ,xy[i]["y"], xy[i]["x"][j],
                xy[i]["y"], height, width[j], tran_time
                    ,delay_time,
                `.x_rows:nth-child(${location[i]})`, msg
                    ,false)
        });
    })
    return_delay = return_delay + tran_time;
    return return_delay;
}

function tbl_mid_cord(c1, c2, xy = true) {
    right = d3.select(`.${c2}_tbl>.${c2}_rows:last-child`)
        .select("rect");
    right_x = parseInt(right.attr("x"));
    right_y = parseInt(right.attr("y")) +
        parseInt(right.attr("height"));
    left = d3.select(`.${c1}_tbl`)
        .select(`.${c1}_cols_last`)
        .select("rect");
    left_y = parseInt(left.attr("y"));
    if (xy === true) {
        let right_rect = d3.select(".y_rows")
            .selectAll(".y_cols:not(.y_cols_key)")
            .nodes();
        let right_w = 0;
        right_rect.forEach(function (d, i) {

            cur_rect = d3.select(d).select("rect");
            right_w = right_w + parseFloat(cur_rect
                .attr("width"));
        })
        left_x = parseInt(left.attr("x")) + parseInt(left
            .attr("width")) + right_w;
    } else {
        left_x = parseInt(left.attr("x")) + parseInt(left
            .attr("width"));
    }

    return ({
        x: (right_x + left_x) / 2,
        y: (right_y + left_y) / 2
    });
}

function get_newcol_xy(initial_prep, totbl_topright
    ,fromtbl_nkey_wd, height) {
    totbl_topright[1] = totbl_topright[1] + height;
    let new_x = [totbl_topright[0]];
```

```
    fromtbl_nkey_wd.map((d, i) => {
        if (i > 0) {
            new_x.push(new_x[i - 1] + fromtbl_nkey_wd[i - 1]);
        }
    });
    let s_cnt = -1;
    let store_obj = new Object();
    Object.keys(initial_prep).forEach(function (d, i, k) {
        cur_data = initial_prep[d];
        if (i === 0) {
            cur_store = [{
                x: new_x,
                y: totbl_topright[1]
            }]
        } else {
            cur_store = new Array();
            for (var j = 0; j < cur_data.length; j++) {
                if (j > 0) {
                    s_cnt++;
                }
                cur_store.push({
                    x: new_x,
                    y: totbl_topright[1] +
                        height * (cur_data[j]["row"] +
                            s_cnt)
                })
            }
        }
        store_obj[i + 1] = cur_store;
    });
    return store_obj;
}

function get_newcol_xyw_cn(to_tbl_id, from_tbl_id) {
    let new_width = new Array()
    let new_xy = tbl_tr_cord(to_tbl_id);
    let new_x = [new_xy[0]];
    let new_y = parseFloat(d3.select(`.${to_tbl_id}_tbl`)
        .select("rect").attr("y"));
    let nodes2move = d3.select(`.${from_tbl_id}_rows`)
        .selectAll(`.${from_tbl_id}_cols
            :not(.${from_tbl_id}_cols_key)`)
        .nodes();
    nodes2move.forEach(function (d, i) {
        cur_rect = d3.select(d).select("rect");
        new_width.push(parseFloat(cur_rect.attr("width")));
        if (i > 0) {
            new_x.push(new_x[i - 1] +
                parseFloat(d3.select(nodes2move[i - 1])
                    .select("rect")
                    .attr("width")))
```

```
        }
    })
    return {
        x: new_x,
        y: new_y,
        width: new_width
    };
}

function tbl_tr_cord(c) {
    tr_rect = d3.select(`.${c}_tbl`)
        .select(`.${c}_cols_last > rect`);
    x = parseFloat(tr_rect.attr("x"));
    y = parseFloat(tr_rect.attr("y"));
    wd = parseFloat(tr_rect.attr("width"));
    return [x + wd, y];
}

function tbl_bl_cord(c) {
    last_row = d3.select(".x_tbl")
        .select(".x_rows:last-of-type");
    first_rect = last_row.select("rect");
    return {
        x: parseFloat(first_rect.attr("x")),
        y: parseFloat(first_rect.attr("y")) +
        parseFloat(first_rect.attr("height"))
    }
}

function msg_box(x, y, width = null, height = null, msg
    ,start_time = 0, tran_time = 1000
        ,pause_time = 1000, center = true) {
    if (width == null) {
        d3.select(".x_rows")
            .selectAll("rect")
            .each(function (d, i) {
                cur_width = d3.select(this).attr("width");
                if (i == 0) {
                    width = cur_width;
                } else {
                    if (cur_width < width) {
                        width = cur_width;
                    }
                }
            });
        width = width * 3;
    }
    if (height == null) {
        height = parseFloat(d3.select("rect")
            .attr("height")) * 2;
    }
```

```
    if (center === true) {
        x = x - width / 2;
        y = y - height / 2;
    }
    let txt_x = x + width / 2;
    let txt_y = y + height / 2;
    let txt_fontsize = d3.select("text")
        .style("font-size");
    msgbox = d3.select("svg")
        .append("g")
        .attr("class", "msgbox");
    if (msg == "gettimeonly") {
        msgbox
            .style("opacity", 0)
    }
    msgbox.append("rect")
        .transition()
        .delay(start_time)
        .duration(tran_time)
        .attr("x", x)
        .attr("y", y)
        .attr("width", width)
        .attr("height", height);
    msgbox.append("text")
        .attr("x", txt_x)
        .attr("y", txt_y)
        .style("font-size", 0)
        .transition()
        .delay(start_time)
        .duration(tran_time)
        .style("font-size", txt_fontsize)
        .text(msg);
    delay_time = start_time + tran_time;
    msgbox
        .transition()
        .delay(delay_time)
        .duration(pause_time)
        .on("end", function () {
            d3.select(this)
                .remove();
        })
    return delay_time + pause_time;
}

function na_rects(x, y, width, height, location = null
    ,tran_time = 1000, delay_time = 0) {
    let svg = d3.select("svg");
    svg
        .append("rect")
        .attr("class", "na_rects")
        .attr("x", x)
```

```
        .attr("y", y)
        .attr("width", 0)
        .attr("height", 0)
        .transition()
        .duration(tran_time)
        .delay(delay_time)
        .attr("width", width)
        .attr("height", height)
        .on("end", function () {
            base = this;
            d3.select(location)
                .insert(function () {
                    return base;
                })
        });
    svg
        .append("text")
        .attr("class", "na_rects")
        .attr("x", x + 0.5 * width)
        .attr("y", y + 0.5 * height)
        .text("NA")
        .style("font-size", 0)
        .transition()
        .delay(delay_time)
        .duration(tran_time)
        .style("font-size", height * 0.55)
        .on("end", function () {
            base = this;
            d3.select(location)
                .insert(function () {
                    return base;
                })
        });;
}

function keycol_anim(to_tbl, from_tbl, speed = 1
    ,start_time = 0) {
    let keypath_tran_time = 1500 / speed;
    let key_opacity_tran_time = 1000 / speed;
    let rectpulse_tran_time = 1500 / speed;
    let join_tran_time = 2000 / speed;
    let delay_time = start_time;
    let key_name = d3.select(`.${to_tbl}_cols_key`)
        .select("text").text();
    let to_keycol_node = d3.select(`.${to_tbl}_cols_key`)
        .node();
    let from_keycol_node = d3.select(`.${from_tbl}_cols_key`)
        .node();
    let to_keycol_node2 = d3.select(`.${to_tbl}_cols_key`)
        .nodes();
    let from_keycol_node2 = d3.select(`.${from_tbl}_cols_key`)
```

```
        .nodes();
    let to_keyrect_sel = d3.select(to_keycol_node)
        .select("rect");
    let from_keyrect_sel = d3.select(from_keycol_node)
        .select("rect");
    let to_keytext_sel = d3.select(to_keycol_node)
        .select("text");
    let to_nkeycol_nodes = d3.select(`.${to_tbl}_rows`)
        .selectAll(`.${to_tbl}_cols
            :not(.${to_tbl}_cols_key)`)
        .nodes();
    let from_nkeycol_nodes = d3.select(`.${from_tbl}_rows`)
        .selectAll(`.${from_tbl}_cols
            :not(.${from_tbl}_cols_key)`)
        .nodes();
    let to_nkeycol_nodes2 =
        d3.selectAll(`.${to_tbl}_cols
            :not(.${to_tbl}_cols_key)`)
        .nodes();
    let from_nkeycol_nodes2 =
        d3.selectAll(`.${from_tbl}_cols
            :not(.${from_tbl}_cols_key)`)
        .nodes();
    let height = parseFloat(to_keyrect_sel.attr("height"));
    let keylink_nodes = new Array();
    to_keyrect_sel.nodes().forEach(function (d, i) {
        one = [
            parseFloat(to_keyrect_sel.attr("x")) +
            parseFloat(to_keyrect_sel.attr("width")) / 2,
            parseFloat(to_keyrect_sel.attr("y"))
        ];
        two = [one[0], height * 0.6];
        three = [
            parseFloat(from_keyrect_sel.attr("x")) +
            parseFloat(from_keyrect_sel.attr("width")) / 2,
            two[1]
        ];
        four = [three[0], one[1]];
        draw_path = `M ${one} ${two} ${three} ${four}`;
        path_sel = d3.select("svg")
            .append("path")
            .attr("class", "key_col_link")
            .attr("d", draw_path)
            .attr("fill", "none")
            .attr('fill-opacity', 0);
        keylink_nodes.push(path_sel.node())
    })
    d3.selectAll(to_nkeycol_nodes2)
        .transition()
        .delay(delay_time)
        .duration(key_opacity_tran_time)
```

```
        .style("opacity", 0.5);
    d3.selectAll(from_nkeycol_nodes2)
        .transition()
        .delay(delay_time)
        .duration(key_opacity_tran_time)
        .style("opacity", 0.5);
    delay_time = delay_time + key_opacity_tran_time;
    keylink_nodes.forEach(function (d, i) {
        length = d.getTotalLength();
        d3.select(d)
            .attr("stroke-dasharray", length + " " + length)
            .attr("stroke-dashoffset", length)
            .transition()
            .delay(delay_time)
            .duration(keypath_tran_time)
            .attr("stroke-dashoffset", 0)
    });
    d3.select("svg")
        .append("text")
        .attr("id", "joinby_text")
        .attr("x", (one[0] + four[0]) / 2)
        .attr("y", one[1])
        .style("opacity", 0)
        .style("font-size", d3.select("text")
            .style("font-size"))
        .style("alignment-baseline", "hanging")
        .text(`Joining by ${key_name}`)
        .transition()
        .duration(keypath_tran_time)
        .delay(delay_time)
        .style("opacity", 1)
        .on("end", function () {
            d3.select(this)
                .remove();
        })
    delay_time = delay_time + keypath_tran_time;
    db_pulse_rect(from_keyrect_sel.nodes(), "yellow"
        ,rectpulse_tran_time, delay_time)
    db_pulse_rect(to_keyrect_sel.nodes(), "yellow"
        ,rectpulse_tran_time, delay_time)
    delay_time = delay_time + rectpulse_tran_time;
    let new_xyd = get_newcol_xyw_cn("x", "y")
    from_nkeycol_nodes.forEach(function (d, i) {
        movexy_cell(d, new_xyd["x"][i], new_xyd["y"]
        ,new_xyd["x"][i],
            new_xyd["y"], height, new_xyd["width"][i]
                ,join_tran_time, delay_time,
            ".x_rows", false)
    })
    delay_time = delay_time + join_tran_time;
    d3.selectAll(to_nkeycol_nodes2)
```

```
        .transition()
        .delay(delay_time)
        .duration(key_opacity_tran_time)
        .style("opacity", 1);
    d3.selectAll(from_nkeycol_nodes2)
        .transition()
        .delay(delay_time)
        .duration(key_opacity_tran_time)
        .style("opacity", 1);
    delay_time = delay_time + key_opacity_tran_time;
    return d3.select("svg")
        .transition()
        .delay(delay_time);
}

function shift_row_down(tbl_id, tbl_sel, row2shift
    ,shiftdown_times, height,
    tran_time, delay_time) {
    tbl_sel.selectAll(`.${tbl_id}_rows:nth-child
        (n+${row2shift + 1}`)
        .each(function (d, i) {
            cur_rect_y = d3.select(this)
                .selectAll("rect")
                .attr("y");
            cur_text_y = d3.select(this)
                .selectAll("text")
                .attr("y");
            d3.select(this)
                .selectAll("rect")
                .transition()
                .duration(tran_time)
                .delay(delay_time)
            d3.select(this)
                .selectAll("text")
                .transition()
                .duration(tran_time)
                .delay(delay_time)
                .attr("y", parseFloat(cur_text_y) +
                    shiftdown_times * height);
        });
    tbl_sel.select(`.${tbl_id}_rows:nth-child(${row2shift}`)
        .transition()
        .delay(delay_time)
        .on("end", function () {
            for (var j = shiftdown_times; j >= 1; j--) {
                cur_row = d3.select(this).clone(true);
                cur_row.style("opacity", 0);
                cur_row.transition()
                    .duration(tran_time / 2)
                    .style("opacity", 1);
                cur_rect_y = parseFloat(cur_row.select("rect")
```

```
                    .attr("y"));
                cur_text_y = parseFloat(cur_row.select("text")
                    .attr("y"));
                cur_col = cur_row.selectAll(`.${tbl_id}_cols`);
                cur_col.selectAll("rect")
                    .transition()
                    .duration(tran_time * 0.7)
                    .attr("y", cur_rect_y + height * j);
                cur_col.selectAll("text")
                    .transition()
                    .duration(tran_time * 0.7)
                    .attr("y", cur_text_y + height * j);
            }
        })
    return delay_time + tran_time;
}

function iso_tbl(join_type = "left", height, speed = 1
    ,start_time = 0) {
    let tran_time = 1500 / speed;
    let delay_time = start_time;
    let to_tbl = "x";
    let from_tbl = "y";
    if (join_type === "right") {
        let to_tbl = "y";
        let from_tbl = "x";
    }
    let svg_wh = {
        height: parseFloat(d3
            .select("svg").attr("height")),
        width: parseFloat(d3
            .select("svg").attr("width"))
    }
    let to_tbl_tl = {
        x: parseFloat(d3.select
            (`.${to_tbl}_cols > rect`)
                .attr("x")),
        y: parseFloat(d3.select
            (`.${to_tbl}_cols > rect`)
                .attr("y"))
    }
    let to_tbl_width = 0;
    d3.select(`.${to_tbl}_rows`).selectAll("rect").nodes()
        .forEach(function (d, i) {
            to_tbl_width += parseFloat(d3.select(d)
                .attr("width"));
        });
    let to_tbl_height = height *
    parseInt(d3.selectAll(".x_rows").nodes().length);
    d3.selectAll(`.${to_tbl}_rows:not(.remove_row)`)
        .nodes().length;
```

```
    let translate_xy = {
        x: svg_wh["width"] / 2 - to_tbl_width /
            2 - to_tbl_tl["x"],
        y: svg_wh["height"] / 2 - to_tbl_height /
            2 - to_tbl_tl["y"]
    };
    d3.selectAll(`.${from_tbl}_tbl, path, .remove_row`)
        .transition()
        .delay(delay_time)
        .duration(tran_time)
        .style("opacity", 0)
        .on("end", function () {
            d3.select(this)
                .remove();
        });
    delay_time = delay_time + tran_time;
    d3.select(`.${to_tbl}_tbl`)
        .transition()
        .delay(delay_time)
        .duration(tran_time)
        .attr("transform", `translate(${translate_xy["x"]},
        ${translate_xy["y"]})`);
    d3.selectAll(`.${to_tbl}_rows:not(.remove_row)`).nodes()
        .forEach(function (d, i) {
            d3.select(d)
                .selectAll("rect")
                .transition()
                .delay(delay_time)
                .duration(tran_time)
                .attr("y", to_tbl_tl["y"] + height * i);
            d3.select(d)
                .selectAll("text")
                .transition()
                .delay(delay_time)
                .duration(tran_time)
                .attr("y", to_tbl_tl["y"] + height * i
                    + height / 2);
        })
    d3.select(`.${to_tbl}_rows`)
        .selectAll("text")
        .transition()
        .delay(delay_time)
        .duration(tran_time)
        .style("font-weight", "bold");

    delay_time = delay_time + tran_time;

    return d3.select("svg")
        .transition().delay(delay_time);
}
```

```
function draw_table(input, x_start, y_start, x_cord, width
    ,height, name, key_col) {
    let y_cord = 0 - height;
    let tbl_name = `${name}_tbl`;
    let row_name = `${name}_rows`
    let col_name = `${name}_cols`
    let key_name = d3.keys(input);
    let new_x_cord = x_cord.map((d, i) => d + x_start);
    let new_y_cord = Array();
    d3.select("svg")
        .append("g")
        .attr("class", tbl_name)
        .selectAll("g")
        .data(input)
        .enter()
        .append("g")
        .attr("class", row_name)
        .each(function (d, i) {
            var header = d3.select(this);
            new_y_cord.push(height * i + y_start);
            d3.keys(d).forEach(function (key, j) {
                header
                    .append("rect")
                    .attr("width", width[j])
                    .attr("height", height)
                    .attr("x", new_x_cord[j])
                    .attr("y", new_y_cord[i])
                    .style("fill", "white");
            });
        });
    d3.selectAll(`.${row_name}`)
        .selectAll("rect").each(function (d, i) {
            var el = this;
            d3.select(el.parentNode)
                .insert("g")
                .attr("class", col_name)
                .append(function () {
                    return el;
                });
        });
    var key_rect = d3.selectAll(`.${col_name}:nth-of-type
        (${key_col})`).nodes();
    d3.selectAll(key_rect)
        .classed(`${col_name}_key`, true)
        .classed("key_col", true);
    var last_rect = d3.selectAll
        (`.${col_name}:last-of-type`).nodes();
    d3.selectAll(last_rect)
        .classed(`${col_name}_last`, true)
        .classed("last_col", true);
    d3.selectAll(`.${row_name}`)
```

```
        .each(function (d, i) {
        input_i = input[i];
        input_i_key = d3.keys(input_i);
        d3.select(this).selectAll(`.${col_name}`)
            .each(function (d, j) {
            d3.select(this)
                .append("text")
                .text(input_i[input_i_key[j]])
                .attr("x", new_x_cord[j] + width[j] * 0.5)
                .attr("y", new_y_cord[i] + height * 0.5)
                .style("font-size", height * 0.5);
        })
    });
    d3.select(`.${row_name}`).selectAll("text")
        .style("font-size", height * 0.5);
    return ({
        x: new_x_cord,
        y: new_y_cord
    });
};

function join_anim(data, speed = 1, join_type = "left"
    ,gray_out = true) {
    if (join_type == "left" || join_type == "complete") {
        return_delay = join_anim_left(data, speed,
            join_type = "left", gray_out);
    } else if (join_type = "inner") {
        return_delay = join_anim_inner(data, speed,
            join_type = "inner", gray_out);
    }
    return d3.select("svg")
        .transition().delay(return_delay);
}

function join_anim_left(data, speed = 1, join_type = "inner"
    ,gray_out = true) {
    let og_xtbl = d3.select(".x_tbl");
    let left_kcol_node = d3.select(".x_tbl")
        .selectAll(".x_cols_key").nodes();
    let right_kcol_node = d3.select(".y_tbl")
        .selectAll(".y_cols_key").nodes();
    let left_row_nodes = d3.selectAll(".x_rows").nodes();
    let right_row_nodes = d3.selectAll(".y_rows").nodes();
    let xy_tbl_cord = tbl_mid_cord("x", "y");
    let height = parseFloat(d3.select("rect").attr("height"));
    let xtbl_topright = tbl_tr_cord("x");
    let ytbl_nkey_wd = nkeycol_width("y");
    let new_xy = get_newcol_xy(data, xtbl_topright
        ,ytbl_nkey_wd, height);
    let shift_cnt = 1;
    let shift_adjy = 0;
```

```
    let line_tran_time = 1200 / speed;
    let line_tran_time2 = 2400 / speed;
    let shiftdown_time = 1500 / speed;
    let join_tran_time = 1500 / speed;
    let na_action_time = 1000 / speed;
    let gray_time = 200 / speed;
    let delay_time = 0;

    let r_link_rect = {};
    d3.keys(data).forEach(function (d, i) {
        let cur_r_link_rect = new Array();
        if (data[d].length > 1) {
            data[d].forEach(function (d2, i2) {
                cur_r_link_rect.push(d3.select
                    (right_kcol_node[d2["dest"]])
                    .select("rect").node());
                r_link_rect[i] = cur_r_link_rect;
            })
        } else {
            if (data[d][0]["dest"] == -1) {
                cur_r_link_rect = -1;
            } else {
                cur_r_link_rect = d3.select
                    (right_kcol_node[data[d][0]["dest"]])
                    .select("rect").node();
            }
        }
        r_link_rect[i] = cur_r_link_rect;
    })
    d3.keys(data).forEach(function (d, i) {
        d3.select("svg")
            .transition()
            .delay(delay_time)
            .on("end", function () {
                msg = {
                    msg: data[d][0].msg
                }
                ind = data[d][0]["row"];
                cur_left_rect = d3.select
                    (left_kcol_node[ind])
                    .select("rect").node();
                cur_left_rect_xy = rect_mid_cord
                    (cur_left_rect)[0];
                cur_left_rect_xy["y"] = cur_left_rect_xy["y"];
                if (msg["msg"] != undefined) {
                    cur_linetime = line_tran_time2;
                } else {
                    cur_linetime = line_tran_time;
                }
                return_delay = link_rect_line2(cur_left_rect
                    ,cur_left_rect_xy, r_link_rect[i], "right",
```

```
                cur_linetime, 0, msg, removeall = true)
            data[d].forEach(function (d2, i2) {
                cur_rows2move = d3.select
                    (right_row_nodes[d2["dest"]])
                    .selectAll(".y_cols:not(.y_cols_key)")
                        .nodes();
                cur_rows2move.forEach(function (d33, i3) {
                    cur_rectwidth = d3.select(d33)
                        .select("rect").attr("width");
                    movexy_cell(cur_rows2move[i3]
                        ,new_xy[1][0]["x"][i3]
                            ,new_xy[1][0]["y"]
                            + shift_adjy * height
                                ,new_xy[1][0]["x"][i3],
                        new_xy[1][0]["y"] + shift_adjy
                            * height, height,
                                cur_rectwidth,
                                join_tran_time,
                                return_delay,
                    `.x_rows:nth-child(${data[d][i2]["row"]
                                + shift_cnt})`, {
                                msg: undefined
                            }, false);
                    })
                    shift_adjy = shift_adjy + 1

            })
            if (data[d].length > 1) {
                shift_row_down("x", og_xtbl
                    ,data[d][0]["row"] + shift_cnt
                        ,data[d].length - 1, height,
                            shiftdown_time
                            ,return_delay);
                shift_cnt = shift_cnt + data[d].length - 1;
            }
        })
    if (data[d][0].msg != undefined) {
        delay_time = delay_time + line_tran_time2
            + join_tran_time + 200 / speed;
    } else {
        delay_time = delay_time + line_tran_time
            + join_tran_time + 200 / speed;
    }
    if (data[d][0]["dest"] == -1) {
        d3.select("svg")
            .transition()
            .delay(delay_time)
            .on("end", function () {
                new_xy[d][0]["x"].forEach(function
                    (d2, i2) {
                    na_rects(d2, new_xy[d][0]["y"]
```

```
                                , ytbl_nkey_wd[i2], height,
                                `.x_rows:nth-child(${
                                    data[d][0]["row"]
                                    + shift_cnt})`,
                                    na_action_time, 0);
                        })
                    })
                delay_time = delay_time + na_action_time;
            } else {
                data[d].forEach(function (d2, i2) {
                    if (d2["dest"] != -1) {
                        d3.select(right_row_nodes[d2["dest"]])
                            .transition()
                            .delay(delay_time)
                            .duration(gray_time)
                            .style("opacity", 0.3);
                    }
                })
                delay_time = delay_time + gray_time;
            }
        })
        return delay_time;
}

function comjoin_final(data, speed = 1, start_time = 0) {
        let row2move = data.row2move;
        let col_ind = data.col_ind;
        if (row2move.length < 0) {
            return 0;
        }
        let tran_time = 1500 / speed;
        let lineqm_time = tran_time / 2;
        let na_time = (tran_time / 1.5) / speed;
        let delay_time = start_time;
        let num_xrows = d3.selectAll(".x_rows").nodes().length;
        let xtbl_bl = tbl_bl_cord("x");
        let xtbl_tr = tbl_tr_cord("x");
        let height = parseFloat(d3.select("rect").attr("height"));
        let last_xrow_rects = d3.select(".x_rows:last-of-type")
            .selectAll("rect")
            .nodes();
        let og_xcols = d3.select(".x_rows")
            .selectAll(".x_cols")
            .nodes();
        let newcols_ind = og_xcols.length + 1;
        let xcol_key_ind = d3.select(".x_rows")
        let new_x = new Array();
        let new_width = new Array();
        let msg_tran_time = 1500 / speed;
        let msg_pause_time = 1500 / speed;
        col_ind.forEach(function (d, i) {
```

```
        new_x.push(parseFloat
            (d3.select(last_xrow_rects[d - 1]).attr("x")));
        new_width.push(parseFloat
            (d3.select(last_xrow_rects[d - 1]).attr("width")));
    })
    let new_y = new Array();
    row2move.forEach(function (d, i) {
        if (i === 0) {
            new_y.push(xtbl_bl["y"]);
        } else {
            new_y.push(new_y[i - 1] + height);
        }
    });
    y_nkey_cols = d3.select(".y_rows")
        .selectAll(".y_cols:not(.y_cols_key)")
        .nodes();
    if (y_nkey_cols.length > 0) {
        na_width = new Array();
        na_x = new Array();
        y_nkey_cols.forEach(function (d, i) {
            na_width.push(parseFloat
                (d3.select(d).select("rect").attr("width")));
            if (i === 0) {
                na_x.push(xtbl_tr[0]);
            } else {
                na_x.push(na_x[i - 1] + na_width[i - 1]);
            }
        });
    }
    let rows2move = new Array();
    row2move.forEach(function (d, i) {
        cur_sel = d3.select(".y_tbl")
            .select(`.y_rows:nth-child(${d + 1})`)
        rows2move.push(cur_sel.node());
        cur_rect = cur_sel.select(".y_cols_key")
            .select("rect");
        new_time = link_rect_line(cur_rect.node()
            ,rect_mid_cord(cur_rect.node())[0], -1
                ,"left", tran_time, delay_time, false);
        d3.select(".x_tbl")
            .append("g")
            .attr("class", "x_rows");
    });
    delay_time = delay_time + new_time;
    d3.selectAll(".qm2")
        .transition()
        .delay(delay_time)
        .duration(lineqm_time)
        .style("font-size", 0)
        .on("end", function () {
            d3.select(this)
```

```
                .remove();
        });
    d3.selectAll("line")
        .transition()
        .delay(delay_time)
        .duration(lineqm_time)
        .style("opacity", 0)
        .on("start", function () {
            msg_xy = tbl_mid_cord("x", "y");
            new_time = msg_box(parseFloat(msg_xy["x"])
                ,parseFloat(msg_xy["y"]), null, null,
                "Move across unused rows"
                    ,0, msg_tran_time, msg_pause_time, true);
        })
        .on("end", function () {
            d3.select(this)
                .remove();
            d3.select(".y_tbl")
                .selectAll("rect")
                .style("fill", null);
        });
    delay_time = delay_time + lineqm_time + new_time;
    num_xrows2 = num_xrows;
    d3.selectAll("svg")
        .transition()
        .delay(delay_time)
        .on("end", function () {
            rows2move.forEach(function (d, i) {
                num_xrows2++;
                inside_cols = d3.select(d)
                    .selectAll(".y_cols").nodes();
                inside_cols.forEach(function (d2, i2) {
                    movexy_cell(d2, new_x[i2], new_y[i]
                        ,new_x[i2], new_y[i], height
                            ,new_width[i2], tran_time, 0,
                        `.x_rows:nth-child(${num_xrows2})`
                            ,false)
                })
            });
        })
    delay_time = delay_time + tran_time;
    if (last_xrow_rects.length > col_ind.length) {
        let x_nkey_last_rects = new Array();
        let seq_n = Array.from(Array(last_xrow_rects.length)
            ,(x, index) => index + 1);
        a_outersect(seq_n, col_ind).forEach(function (d, i) {
            x_nkey_last_rects.push(last_xrow_rects[d - 1]);
        });
        let na_x = [];
        let na_y = new_y;
        let na_width = [];
```

```
        let naregion_height = row2move.length * height;
        x_nkey_last_rects.forEach(function (d, i) {
            na_x.push(parseFloat(d3.select(d).attr("x")));
            na_width.push(parseFloat(d3.select(d)
                .attr("width")));
        })
        x_nkey_last_rects.forEach(function (d, i) {
            na_region = d3.select("svg")
                .append("rect");
            na_region
                .attr("x", na_x[i])
                .attr("y", na_y[0])
                .attr("width", na_width[i])
                .attr("height", naregion_height)
                .attr("class", "na_region")
                .style("opacity", 0)
                .style("fill", "red")
                .transition()
                .delay(delay_time)
                .duration(tran_time)
                .style("opacity", 1);
        });
        delay_time = delay_time + tran_time;
        let na_regions = d3.selectAll(".na_region")
            .nodes();
        na_regions.forEach(function (d, i) {
            cur_mid = rect_mid_cord(d)[0];
            d3.select("svg")
                .append("text")
                .text("?")
                .attr("x", cur_mid["x"])
                .attr("y", cur_mid["y"])
                .attr("class", "na_region_txt")
                .style("font-size", 0)
                .transition()
                .delay(delay_time)
                .duration(na_time)
                .style("font-size", parseFloat(d3.select(d)
                    .attr("height")) / 2)
                .on("end", function () {
                    db_pulse_rect(d, "white", na_time, 2, 0);
                    d3.select(d)
                        .transition()
                        .delay(na_time / 2)
                        .duration(na_time / 2)
                        .style("opacity", 0)
                        .on("end", function () {
                            d3.select(this).remove();
                        });
                    d3.selectAll(".na_region_txt")
                        .transition()
```

```
                        .delay(na_time / 2)
                        .duration(na_time / 2)
                        .style("opacity", 0)
                        .on("end", function () {
                            d3.select(this).remove();
                        });
                });
        });
        delay_time = delay_time + 2 * na_time;
        let num_xrows2 = num_xrows;
        x_nkey_last_rects.forEach(function (d, i) {
            row2move.forEach(function (d2, i2) {
                na_rects(na_x[i], new_y[i2], na_width[i]
                    ,height,
                `.x_rows:nth-child(${num_xrows + 1})`,
                    na_time, delay_time);
                num_xrows++;
            });
            num_xrows = num_xrows2;
        });
        delay_time = delay_time + na_time * 2;
    }
    d3.selectAll("line")
        .transition()
        .delay(delay_time)
        .on("end", function () {
            d3.select(this)
                .remove();
            d3.selectAll(".qm2")
                .style("opacity", 0);
        });
    return d3.select("svg")
        .transition().delay(delay_time);
}

function join_anim_inner(data, speed = 1, join_type = "inner"
    ,gray_out = true) {
    let og_xtbl = d3.select(".x_tbl");
    let left_kcol_node = d3.select(".x_tbl")
        .selectAll(".x_cols_key").nodes();
    let right_kcol_node = d3.select(".y_tbl")
        .selectAll(".y_cols_key").nodes();
    let left_row_nodes = d3.selectAll(".x_rows").nodes();
    let right_row_nodes = d3.selectAll(".y_rows").nodes();
    let xy_tbl_cord = tbl_mid_cord("x", "y");
    let height = parseFloat(d3.select("rect").attr("height"));
    let xtbl_topright = tbl_tr_cord("x");
    let ytbl_nkey_wd = nkeycol_width("y");
    let new_xy = get_newcol_xy(data, xtbl_topright
        ,ytbl_nkey_wd, height);
    let shift_cnt = 1;
```

```
let shift_adjy = 0;
let line_tran_time = 1200 / speed;
let line_tran_time2 = 2400 / speed;
let shiftdown_time = 1500 / speed;
let join_tran_time = 1500 / speed;
let na_action_time = 500 / speed;
let gray_time = 200 / speed;
let delay_time = 0;
let temp = 0;
let r_link_rect = {};
d3.keys(data).forEach(function (d, i) {
    let cur_r_link_rect = new Array();
    if (data[d].length > 1) {
        data[d].forEach(function (d2, i2) {
            cur_r_link_rect.push(d3.select
                (right_kcol_node[d2["dest"]])
                    .select("rect").node());
            r_link_rect[i] = cur_r_link_rect;
        })
    } else {
        if (data[d][0]["dest"] == -1) {
            cur_r_link_rect = -1;
        } else {
            cur_r_link_rect = d3.select
                (right_kcol_node[data[d][0]["dest"]])
                .select("rect").node();
        }
    }
    r_link_rect[i] = cur_r_link_rect;
})
d3.keys(data).forEach(function (d, i) {
    d3.select("svg")
        .transition()
        .delay(delay_time)
        .on("end", function () {
            msg = {
                msg: data[d][0].msg
            }
            ind = data[d][0]["row"];
            cur_left_rect = d3.select
                (left_kcol_node[ind])
                .select("rect").node();
            cur_left_rect_xy = rect_mid_cord
                (cur_left_rect)[0];
            cur_left_rect_xy["y"] = cur_left_rect_xy["y"];
            if (msg["msg"] != undefined) {
                cur_linetime = line_tran_time2;
            } else {
                cur_linetime = line_tran_time;
            }
            return_delay = link_rect_line2
```

```
                    (cur_left_rect, cur_left_rect_xy
                        ,r_link_rect[i], "right",
                    cur_linetime, 0, msg, removeall = true)
                data[d].forEach(function (d2, i2) {
                    cur_rows2move = d3.select
                       (right_row_nodes[d2["dest"]]).selectAll
                        (".y_cols:not(.y_cols_key)").nodes();
                    cur_rows2move.forEach(function (d33, i3) {
                        cur_rectwidth = d3.select(d33)
                            .select("rect").attr("width");
                        movexy_cell(cur_rows2move[i3]
                            ,new_xy[1][0]["x"][i3]
                                ,new_xy[1][0]["y"] +
                                shift_adjy * height,
                                new_xy[1][0]["x"][i3],
                            new_xy[1][0]["y"] + shift_adjy *
                            height, height, cur_rectwidth,
                            join_tran_time, return_delay,
                            `.x_rows:nth-child
                                (${data[d][i2]["row"]
                                    + shift_cnt})`, {
                                msg: undefined
                            }, false);
                    })
                    shift_adjy = shift_adjy + 1
                })
                if (data[d].length > 1) {
                    shift_row_down("x",
                        og_xtbl, data[d][0]["row"]
                        + shift_cnt, data[d].length - 1
                            ,height, shiftdown_time,
                            return_delay);
                    shift_cnt = shift_cnt +
                        data[d].length - 1;
                }
            })
        if (data[d][0].msg != undefined) {
            delay_time = delay_time + line_tran_time2
                + join_tran_time + 200 / speed;
        } else {
            delay_time = delay_time + line_tran_time
                + join_tran_time + 200 / speed;
        }
        if (data[d][0]["dest"] == -1) {
            d3.select(`.x_rows:nth-child
                (${data[d][0]["row"] + shift_cnt })`)
                .classed("remove_row", true)
                .transition()
                .delay(delay_time)
                .duration(na_action_time)
                .style("opacity", 0);
```

```
            delay_time = delay_time + na_action_time;
        } else {
            data[d].forEach(function (d2, i2) {
                if (d2["dest"] != -1) {
                    d3.select(right_row_nodes[d2["dest"]])
                        .transition()
                        .delay(delay_time)
                        .duration(gray_time)
                        .style("opacity", 0.3);
                }
            })
            delay_time = delay_time + gray_time;
        }
    })
    return delay_time;
}

function join_finalcheck(data) {
    let all_xrows = d3.selectAll(".x_rows").nodes();
    let cnt = 1;
    let bad_xrows = new Array();
    d3.keys(data).forEach(function (d, i) {
        if (data[d].length > 1) {
            bad_xrows.push(cnt);
        }
        cnt += data[d].length;
    })
    let num_ynkey_col = d3.select(".y_rows")
        .selectAll(".y_cols:not(.y_cols_key)")
        .nodes().length;
    d3.select("svg")
        .transition()
        .delay(250)
        .on("end", function () {
            bad_xrows.forEach(function (d, i) {
                cur_badrect = d3.selectAll
                    (`.x_rows:nth-child(${d + 1}) > rect`)
                        .nodes();
                cur_badtext = d3.selectAll
                    (`.x_rows:nth-child(${d + 1}) > text`)
                        .nodes();
                iterate_times = cur_badrect.length
                    / num_ynkey_col - 1;
                cur_badrect = cur_badrect
                    .slice(num_ynkey_col, cur_badrect.length);
                cur_badtext = cur_badtext
                    .slice(num_ynkey_col, cur_badtext.length);
                for (let j = 1; j <= iterate_times; j++) {
                    for (let k = 0; k < num_ynkey_col; k++) {
                        d3.select(`.x_rows
                            :nth-child(${d + 1 + j})`)
```

```
                            .insert(function () {
                                return cur_badrect[0];
                            })
                        d3.select(`.x_rows
                            :nth-child(${d + 1 + j})`)
                            .insert(function () {
                                return cur_badtext[0];
                            })
                        cur_badrect.shift();
                        cur_badtext.shift();
                    }
                }
            })
        })
    return d3.select("svg").transition()
        .delay(500);
}
```

B.2 Reshaping module

```
function vabline(v) {
  d3.select("svg")
    .append("line")
    .attr("x1", v)
    .attr("x2", v)
    .attr("y1", -9999)
    .attr("y2", 9999)
    .style("stroke-width", "10px")
    .style("fill", "red");
}
function habline(h) {
  d3.select("svg")
    .append("line")
    .attr("x1", -9999)
    .attr("x2", 9999)
    .attr("y1", h)
    .attr("y2", h)
    .style("stroke-width", "10px")
    .style("fill", "red");
}
function get_translation(obj, type) {
  if (type == "node") {
    string = d3.select(obj).attr("transform");
  } else {
    string = obj.attr("transform");
  }
  if (string === null) {
    return ([0, 0]);
  } else {
    string = string.substring(string.indexOf("(") + 1,
      string.indexOf(")")).split(",");
```

```
    return string.map((d, i) => parseInt(d));
  }
}
function clone_everything(selector) {
  var node = d3.select(selector).node();
  return d3.select(node.parentNode.insertBefore(node.
    cloneNode(true), node.nextSibling));
};
function a_intersect(a, b) {
  var t;
  if (b.length > a.length) t = b, b = a, a = t;
  return a.filter(function (e) {
    return b.indexOf(e) > -1;
  });
}
function a_intersect2(a, b) {
  var t;
  if (b.length > a.length) t = b, b = a, a = t;
  return a.filter(function (e) {
    return b.indexOf(e) > -1;
  }).filter(function (e, i, c) {
    return c.indexOf(e) === i;
  });
}
function a_outersect(a, b) {
  return a.filter(x => !b.includes(x))
}
function arr_last(arr) {
  return arr.slice(-1)[0];
}
function arr_nth(arr, n) {
  return arr.filter(function (value, index, ar) {
    index++;
    return (index % n == 0);
  });
}
function arr_sum(arr) {
  let reducer = (accumulator, currentValue)
    => accumulator + currentValue; return arr.reduce(reducer);
}

function arr_scale(arr, sc) {
  arr = arr.map((d, i) => {
    return (sc(d))
  });
  return arr;
}
function parent_setclass(sel, s_class, level) {
  for (var i = 0; i < level; i++) {
    sel = sel.select(function () {
      return this.parentNode;
```

```
    });
    sel.classed(s_class, true);
  }
}
function insertnodeinto(nn, seltxt) {
  d3.select(seltxt)
    .insert(function () {
      return nn;
    })
}
function db_pulse_rect(nodes, col, tran_time = 1000,
    delay_time = 0, keep_highlighted = false) {
  if (nodes.length === undefined) {
    let og_col = d3.select(nodes).style("fill");
    for (var j = 0; j < 4; j++) {
      if (j % 2 === 0) {
        d3.select(nodes)
          .transition()
          .duration(tran_time / 4)
          .delay(delay_time + tran_time / 4 * j)
          .style("fill", col);
      } else {
        d3.select(nodes)
          .transition()
          .duration(tran_time / 4)
          .delay(delay_time + tran_time / 4 * j)
          .style("fill", og_col);
      }
    }
    if (keep_highlighted === true) {
      d3.select(nodes)
        .transition()
        .delay(delay_time + tran_time)
        .style("fill", col);
    }
  } else {
    nodes.forEach(function (d, i) {
      for (var j = 0; j < 4; j++) {
        if (j % 2 === 0) {
          d3.select(d)
            .transition()
            .duration(tran_time / 4)
            .delay(delay_time + tran_time / 4 * j)
            .style("fill", col);
        } else {
          d3.select(d)
            .transition()
            .duration(tran_time / 4)
            .delay(delay_time + tran_time / 4 * j)
            .style("fill", null);
        }
```

```
      }
      if (keep_highlighted === true) {
        d3.select(d)
          .transition()
          .delay(delay_time + tran_time)
          .style("fill", col);
      }
    })
  }
}
function db_pulse_text(nodes, col, tran_time = 1000,
    delay_time = 0, keep_highlighted = false) {
  if (nodes.length === undefined) {
    let og_col = d3.select(nodes).style("fill");
    for (var j = 0; j < 4; j++) {
      if (j % 2 === 0) {
        d3.select(nodes)
          .transition()
          .duration(tran_time / 4)
          .delay(delay_time + tran_time / 4 * j)
          .style("fill", col);
      } else {
        d3.select(nodes)
          .transition()
          .duration(tran_time / 4)
          .delay(delay_time + tran_time / 4 * j)
          .style("fill", og_col);
      }
    }
    if (keep_highlighted === true) {
      d3.select(nodes)
        .transition()
        .delay(delay_time + tran_time)
        .style("fill", col);
    }
  } else {
    nodes.forEach(function (d, i) {
      for (var j = 0; j < 4; j++) {
        if (j % 2 === 0) {
          d3.select(d)
            .transition()
            .duration(tran_time / 4)
            .delay(delay_time + tran_time / 4 * j)
            .style("fill", col);
        } else {
          d3.select(d)
            .transition()
            .duration(tran_time / 4)
            .delay(delay_time + tran_time / 4 * j)
            .style("fill", null);
        }
```

```
      }
      if (keep_highlighted === true) {
        d3.select(d)
          .transition()
          .delay(delay_time + tran_time)
          .style("fill", col);
      }
    })
  }
}
function rect_mid_cord(nodes) {
  let result = new Array();
  if (nodes.length == undefined) {
    x = parseFloat(d3.select(nodes).attr("x"));
    y = parseFloat(d3.select(nodes).attr("y"));
    wd = parseFloat(d3.select(nodes).attr("width"));
    ht = parseFloat(d3.select(nodes).attr("height"));
    result.push({
      x: x + wd / 2,
      y: y + ht / 2
    });
  } else {
    nodes.forEach(function (d, i) {
      x = parseFloat(d3.select(d).attr("x"));
      y = parseFloat(d3.select(d).attr("y"));
      wd = parseFloat(d3.select(d).attr("width"));
      ht = parseFloat(d3.select(d).attr("height"));
      result.push({
        x: x + wd / 2,
        y: y + ht / 2
      });
    })
  }
  return result;
}
function nkeycol_width(id) {
  let temp_nodes = d3.select(`.${id}_rows`)
    .selectAll(`.${id}_cols:not(.${id}_cols_key`)
    .select("rect").nodes();
  let store = new Array()
  temp_nodes.forEach(function (d, i) {
    store.push(parseFloat(d3.select(d).attr("width")));
  })
  return store;
}
function move_text(to_sel, from_node, tran_time, delay_time) {
d3.select(from_node)
    .select("text")
    .classed("moved", true)
    .transition()
    .delay(delay_time)
```

```
    .duration(tran_time)
    .on("start", function (d, i) {
      d3.select(this)
        .clone(true);
    })
    .attr("x", to_sel.attr("x"))
    .attr("y", to_sel.attr("y"))
    .on("end", function (d, i) {
      d3.select(this)
        .remove();
    });
}
function movexy_rect(node, x, y, width, tran_time, delay_time,
  insert_txt = null, remove = false) {

  if (y === null) {
    y = d3.select(node)
      .select("rect")
      .attr("y")
  }
if (insert_txt != null || insert_txt != undefined) {
  node = d3.select(node)
    .select("rect")
    .clone(true)
    .node();
  d3.select(".x_rows")
    .insert(function () {
      return node;
    })
  d3.select(node).classed("moved", true)
    .transition()
    .delay(delay_time)
    .duration(tran_time)
    .attr("x", x)
    .attr("y", y)
    .attr("width", width)
    .on("end", function (d, i) {
      if (remove === true) {
        d3.select(this)
          .remove();
      }
    });
}
function movexy_text(node, x, y, adj_y = null, width, tran_time, delay_time,
  insert_txt = null, remove = false) {
  parent_setclass(d3.select(node), "moved", 1);
  if (y === null) {
    y = d3.select(node)
      .select("text")
      .attr("y");
  }
```

```
  if (adj_y !== null) {
    y = y + 0.5 * adj_y;
  }
  node = d3.select(node)
    .select("text")
    .clone(true);
  node.classed("moved", true)
    .transition()
    .delay(delay_time)
    .duration(tran_time)
    .attr("x", parseInt(x) + 0.5 * width)
    .attr("y", y)
    .on("end", function (d, i) {
      if (remove === true) {
        d3.select(this)
          .remove();
      }
    });
}
function movexy_cell(node, rx, ry, tx, ty, adj_ty = null, width, tran_ti
  location = null, msg = undefined, remove = false) {
  rnode = d3.select(node)
    .select("rect")
    .clone(true)
    .node();
  tnode = d3.select(node)
    .select("text")
    .clone(true)
    .node();
  if (ry === null) {
    ry = d3.select(node)
      .select("rect")
      .attr("y")
  }
  d3.select(rnode)
    .transition()
    .delay(delay_time)
    .duration(tran_time)
    .attr("x", rx)
    .attr("y", ry)
    .attr("width", width)
    .on("end", function (d, i) {
      if (location != null) {
        base = this;
        d3.select(location)
          .insert(function () {
            return base;
          })
      }
    });
  if (ty === null) {
```

```
    ty = d3.select(node)
      .select("text")
      .attr("y");
  }
  if (adj_ty !== null) {
    ty = ty + 0.5 * adj_ty;
  }
  d3.select(tnode)
    .transition()
    .delay(delay_time)
    .duration(tran_time)
    .attr("x", parseFloat(tx) + 0.5 * width)
    .attr("y", ty)
    .on("end", function (d, i) {
      if (location != null) {
        base = this;
        d3.select(location)
          .insert(function () {
            return base;
          })
      }
    });
  return delay_time + tran_time;
}
function tbl_mid_cord(c1, c2, xy = true) {
  right = d3.select(`.${c2}_tbl>.${c2}_rows:last-child`)
    .select("rect");
  right_x = parseInt(right.attr("x"));
  right_y = parseInt(right.attr("y")) + parseInt(right.attr("height"));
  left = d3.select(`.${c1}_tbl`)
    .select(`.${c1}_cols_last`)
    .select("rect");
  left_y = parseInt(left.attr("y"));
  if (xy === true) {
    let right_rect = d3.select(".y_rows")
      .selectAll(".y_cols:not(.y_cols_key)")
      .nodes();
    let right_w = 0;
    right_rect.forEach(function (d, i) {
      cur_rect = d3.select(d).select("rect");
      right_w = right_w + parseFloat(cur_rect.attr("width"));
    })
    left_x = parseInt(left.attr("x")) + parseInt(left.
        attr("width")) + right_w;
  } else {
    left_x = parseInt(left.attr("x")) + parseInt(left.
        attr("width"));
  }
  return ({
    x: (right_x + left_x) / 2,
    y: (right_y + left_y) / 2
```

```
  });
}
function msg_box2(x, y, width = null, height = null, msg,
    start_time = 0, tran_time = 2000, pause_ratio = 0.5,
        center = true) {
 pause_time = tran_time * pause_ratio;
  tran_time = tran_time - pause_time;
  if (width == null) {
    d3.select(".x_rows")
      .selectAll("rect")
      .each(function (d, i) {
        cur_width = d3.select(this).attr("width");
        if (i == 0) {
          width = cur_width;
        } else {
          if (cur_width < width) {
            width = cur_width;
          }
        }
      });
    width = width * 3;
  }
  if (height == null) {
    height = parseFloat(d3.select("rect").attr("height")) * 2;
  }
  if (center === true) {
    x = x - width / 2;
    y = y - height / 2;
  }
  let txt_x = x + width / 2;
  let txt_y = y + height / 2;
  let txt_fontsize = d3.select("text")
    .style("font-size");
  msgbox = d3.select("svg")
    .append("g")
    .attr("class", "msgbox");
  if (msg == "gettimeonly") {
    msgbox
      .style("opacity", 0)
  }
  msgbox.append("rect")
    .transition()
    .delay(start_time)
    .duration(tran_time)
    .attr("x", x)
    .attr("y", y)
    .attr("width", width)
    .attr("height", height);
  msgbox.append("text")
    .attr("x", txt_x)
    .attr("y", txt_y)
```

```
    .style("font-size", 0)
    .transition()
    .delay(start_time)
    .duration(tran_time)
    .style("font-size", txt_fontsize)
    .text(msg);
  delay_time = start_time + tran_time;
  msgbox
    .transition()
    .delay(delay_time)
    .duration(pause_time)
    .on("end", function () {
      d3.select(this)
        .remove();
    })
  return delay_time + pause_time;
}
function animate_line(x1, x2, y1, y2, tran_time = 1000,
    start_time = 0) {
  cur_line = d3.select("svg")
    .append("line")
    .attr("x1", x1)
    .attr("x2", x1)
    .attr("y1", y1)
    .attr("y2", y1);
  cur_line.transition()
    .delay(start_time)
    .duration(tran_time)
    .attr("x2", x2)
    .attr("y2", y2)
    .on("end", function () {
      d3.select(this)
        .remove();
    });
}
function iso_msgbox(x, y, width = null, height = null, msg,
    start_time = 0, tran_time = 2000, isomsg_ratio = 0.5,
        pause_time = 1000, center = true) {
  d3.selectAll("svg > *")
    .transition()
    .delay(start_time)
    .style("opacity", 0.3)
    .transition()
    .delay(tran_time * 0.8)
    .on("end", function () {
      d3.select(this)
        .transition()
        .duration(tran_time * 0.15)
        .style("opacity", 1);
    });
  msg_box2(x, y, width, height, msg, start_time, tran_time, isomsg_ratio, ce
```

```
}
function draw_table(input, x_start, y_start, x_cord, width,
    height, name, key_col) {
  let y_cord = 0 - height;
  let tbl_name = `${name}_tbl`;
  let row_name = `${name}_rows`
  let col_name = `${name}_cols`
  let key_name = d3.keys(input);
  let new_x_cord = x_cord.map((d, i) => d + x_start);
  let new_y_cord = Array();
  d3.select("svg")
    .append("g")
    .attr("class", tbl_name)
    .selectAll("g")
    .data(input)
    .enter()
    .append("g")
    .attr("class", row_name)
    .each(function (d, i) {
      var header = d3.select(this);
      new_y_cord.push(height * i + y_start);
      d3.keys(d).forEach(function (key, j) {
        header
          .append("rect")
          .attr("width", width[j])
          .attr("height", height)
          .attr("x", new_x_cord[j])
          .attr("y", new_y_cord[i])
          .style("fill", "white");
      });
    });
  d3.selectAll(`.${row_name}`)
    .selectAll("rect").each(function (d, i) {
      var el = this;
      d3.select(el.parentNode)
        .insert("g")
        .attr("class", col_name)
        .append(function () {
          return el;
        });
    });
  var key_rect = d3.selectAll(`.${col_name}:nth-of-
    type(${key_col})`).nodes();
  d3.selectAll(key_rect)
    .classed(`${col_name}_piv`, true)
    .classed("piv_col", true);
  var last_rect = d3.selectAll(`.${col_name}:last-of-type`).
    nodes();
  d3.selectAll(last_rect)
    .classed(`${col_name}_last`, true)
    .classed("last_col", true);
```

```
  d3.selectAll(`.${row_name}`).
    each(function (d, i) {
    input_i = input[i];
    input_i_key = d3.keys(input_i);
    d3.select(this).selectAll(`.${col_name}`).each(function (d, j) {
      d3.select(this)
        .append("text")
        .text(input_i[input_i_key[j]])
        .attr("x", new_x_cord[j] + width[j] * 0.5)
        .attr("y", new_y_cord[i] + height * 0.5)
        .style("font-size", height * 0.5)
        .on("end", function () {
          d3.select(this)
            .remove();
        });
    })
  });
  d3.select(`.${row_name}`).selectAll("text").style(
    "font-size", height * 0.5);
  return ({
    x: new_x_cord,
    y: new_y_cord
  });
};
function draw_table_rectonly(input, x_start, y_start, x_cord,
    width, height, name, key_col, pivot_text = false) {
  let y_cord = 0 - height;
  let tbl_name = `${name}_tbl`;
  let row_name = `${name}_rows`
  let col_name = `${name}_cols`
  let key_name = d3.keys(input);
  let new_x_cord = x_cord.map((d, i) => d + x_start);
  let new_y_cord = Array();
  d3.select("svg")
    .append("g")
    .attr("class", tbl_name)
    .selectAll("g")
    .data(input)
    .enter()
    .append("g")
    .attr("class", row_name)
    .each(function (d, i) {
      var header = d3.select(this);
      new_y_cord.push(height * i + y_start);
      d3.keys(d).forEach(function (key, j) {
        header
          .append("rect")
          .attr("width", width[j])
          .attr("height", height)
          .attr("x", new_x_cord[j])
          .attr("y", new_y_cord[i])
```

```
          .style("fill", "white");
      });
    });
  d3.selectAll(`.${row_name}`)
    .selectAll("rect").each(function (d, i) {
      var el = this;
      d3.select(el.parentNode)
        .insert("g")
        .attr("class", col_name)
        .append(function () {
          return el;
        });
    });
  var key_rect = d3.selectAll(`.${col_name}:nth-of-
    type(${key_col})`).nodes();
  d3.selectAll(key_rect)
    .classed(`${col_name}_piv`, true)
    .classed("piv_col", true);
  var last_rect = d3.selectAll(`.${col_name}:last-of-type`).
    nodes();
  d3.selectAll(last_rect)
    .classed(`${col_name}_last`, true)
    .classed("last_col", true);
  d3.selectAll(`.${row_name}`).each(function (d, i) {
    input_i = input[i];
    input_i_key = d3.keys(input_i);
    d3.select(this).selectAll(`.${col_name}`).each(function
        (d, j) {
      if (pivot_text === true) {
        if (j == 0) {
          d3.select(this)
            .append("text")
            .text(input_i[input_i_key[j]])
            .attr("x", new_x_cord[j] + width[j] * 0.5)
            .attr("y", new_y_cord[i] + height * 0.5)
            .style("font-size", height * 0.5)
            .on("end", function () {
              d3.select(this)
                .remove();
            });
        }
      }
    })
  });
  d3.select(`.${row_name}`).selectAll("text").style("font-size",
    height * 0.5);
  return ({
    x: new_x_cord,
    y: new_y_cord
  });
};
```

```
function prepare_gather(start_xy, col_ind, key, value,
    key_width, value_width, height, start_time = 0,
        speed = 1) {
  let tran_time = 2000 / speed;
  let delay_time = start_time;
  anim_header = d3.select("svg")
    .append("g")
    .attr("id", "anim_header");
  anim_header
    .append("text")
    .attr("id", "key_txt1")
    .attr("x", start_xy["x"])
    .attr("y", start_xy["y"])
    .style("opacity", 0)
    .style("text-anchor", "start")
    .style("font-size", height * 0.6)
    .text("Make a new column ");
  key_rect_x = d3.select("#key_txt1").node().
    getComputedTextLength() * 1.025 +
    start_xy["x"];
  key_rect_y = start_xy["y"] - height / 2;
  anim_header.append("g")
    .attr("id", "key_el")
    .append("rect")
    .style("opacity", 0)
    .attr("id", "key_rect")
    .attr("x", key_rect_x)
    .attr("y", key_rect_y)
    .attr("width", key_width)
    .attr("height", height);
  d3.select("#key_el")
    .append("text")
    .attr("id", "key_text")
    .attr("x", key_rect_x + key_width / 2)
    .attr("y", key_rect_y + height / 2)
    .text(key)
    .style("opacity", 0)
    .style("font-size", height * 0.5);
  anim_header.select("#key_txt1")
    .append("tspan")
    .attr("id", "key_txt2")
    .attr("x", key_rect_x + key_width * 1.05)
    .text(" to contain the old column names ")
  d3.select("#key_txt1")
    .transition()
    .delay(start_time)
    .duration(tran_time)
    .style("opacity", 1);
  d3.select("#key_rect")
    .transition()
    .delay(start_time)
```

```
    .duration(tran_time)
    .style("opacity", 1);
  d3.select("#key_text")
    .transition()
    .delay(start_time)
    .duration(tran_time)
    .style("opacity", 1);
  delay_time = delay_time + tran_time;
  key_rect_x = key_rect_x + d3.select("#key_txt2").node().
    getComputedTextLength() + key_width * 1.1;
  adj_width = 0;
  adj_text_x = 0;
  col_ind.forEach(function (d, i) {
    cur_head_move = d3.select(".o_rows")
      .selectAll(`.o_cols:nth-child(${d})`);
    cur_move_rect = cur_head_move.select("rect")
      .clone(true);
    cur_move_text = cur_head_move.select("text")
      .clone(true);
    insertnodeinto(cur_move_rect.node(), "#anim_header");
    insertnodeinto(cur_move_text.node(), "#anim_header");
    cur_move_rect
      .transition()
      .delay(delay_time)
      .duration(tran_time)
      .attr("x", key_rect_x + adj_width * 1.05)
      .attr("y", key_rect_y);
    adj_text_x = parseFloat(cur_head_move.select("rect").
        attr("width"));
    cur_move_text
      .transition()
      .delay(delay_time)
      .duration(tran_time)
      .attr("x", key_rect_x + adj_width * 1.05 + adj_text_x / 2)
      .attr("y", key_rect_y + height / 2);
    adj_width = adj_text_x;
  })
  delay_time = delay_time + tran_time;
  start_xy["y"] = start_xy["y"] + height * 1.2;
  anim_header
    .append("text")
    .attr("id", "value_txt")
    .attr("x", start_xy["x"])
    .attr("y", start_xy["y"])
    .style("opacity", 0)
    .style("text-anchor", "start")
    .style("font-size", height * 0.6)
    .text("Make a new column ");
  value_rect_x = d3.select("#value_txt").node().
    getComputedTextLength() * 1.025 +
    start_xy["x"];
```

```
  value_rect_y = start_xy["y"] - height / 2;
  anim_header.append("g")
    .attr("id", "value_el")
    .append("rect")
    .style("opacity", 0)
    .attr("id", "value_rect")
    .attr("x", value_rect_x)
    .attr("y", value_rect_y)
    .attr("width", value_width)
    .attr("height", height);
  d3.select("#value_el")
    .append("text")
    .style("opacity", 0)
    .attr("id", "value_text")
    .attr("x", value_rect_x + value_width / 2)
    .attr("y", value_rect_y + height / 2)
    .text(value)
    .style("font-size", height * 0.5);
  anim_header.select("#value_txt")
    .append("tspan")
    .attr("id", "key_txt2")
    .attr("x", value_rect_x + value_width * 1.05)
    .text(" to store the values from the old columns.")
  d3.select("#value_txt")
    .transition()
    .delay(delay_time)
    .duration(tran_time)
    .style("opacity", 1);
  d3.select("#value_rect")
    .transition()
    .delay(delay_time)
    .duration(tran_time)
    .style("opacity", 1);
  d3.select("#value_text")
    .transition()
    .delay(delay_time)
    .duration(tran_time)
    .style("opacity", 1);
  delay_time = delay_time + tran_time;
  return d3.select("svg")
    .transition().delay(delay_time);
}
function make_rtbl_g(data) {
 d3.select("svg")
    .append("g")
    .attr("class", "r_tbl");
  d3.keys(data).forEach(function (d, i) {
    cur_row = d3.select(".r_tbl")
      .append("g");
    cur_row.attr("class", "r_rows");
  })
```

```
}
function gather_anim_pivot(tbl_mid_xy, current_chunk,
    piv_ind_old = 1, piv_ind_new = 1, speed = 1,
        start_time = 0, msg = false, first = false) {
  let delay_time = start_time;
  let tran_time = 2000 / speed;
  let tran_time2 = 3000 / speed;
  let iso_time = 4000 / speed;
  let pause_time = 1500 / speed;
  let height = parseFloat(d3.select("rect").attr("height"));
  let o_pivot_cols = d3.selectAll(".o_rows")
    .selectAll(`.o_cols:nth-child(${piv_ind_old})`)
    .nodes();
  let o_pivot_header = o_pivot_cols[0];
  let o_pivot_header_rect = d3.select(o_pivot_header)
    .select("rect").node();
  o_pivot_cols.shift();
  let o_pivot_els = o_pivot_cols;
  let r_pivot_header = d3.select(".r_rows")
    .selectAll(`.r_cols:nth-child(${piv_ind_old})`)
    .node();
  let r_pivot_header_rect = d3.select(r_pivot_header)
    .select("rect").node();
  let r_pivot_els = d3.selectAll(current_chunk)
    .selectAll(`.r_cols:nth-child(${piv_ind_new})`)
    .nodes();
  if (first === true) {
    d3.selectAll(".r_rows")
      .transition()
      .delay(delay_time)
      .duration(tran_time)
      .style("opacity", 1);
    delay_time = delay_time + tran_time;
  }
  if (msg === true) {
    iso_msgbox(tbl_mid_xy["x"], tbl_mid_xy["y"], height * 10,
        height, "Start wide to long transformation",
            delay_time, iso_time, 0.5, pause_time);
    delay_time = delay_time + iso_time;
  }
  db_pulse_rect(o_pivot_header_rect, "yellow", tran_time,
    delay_time);
  db_pulse_rect(r_pivot_header_rect, "yellow", tran_time,
    delay_time);
  old_xy = rect_mid_cord(o_pivot_header_rect)[0];
  new_xy = rect_mid_cord(r_pivot_header_rect)[0];
  animate_line(old_xy["x"], new_xy["x"], old_xy["y"],
    new_xy["y"], tran_time, delay_time);
  delay_time = delay_time + tran_time;
  o_pivot_els.forEach(function (d, i) {
    new_rect = d3.select(r_pivot_els[i])
```

```
      .select("rect");
    new_x = parseFloat(new_rect.attr("x"));
    new_y = parseFloat(new_rect.attr("y"));
    new_wd = parseFloat(new_rect.attr("width"));
    d3.select(d).select("rect")
      .clone(true)
      .transition()
      .delay(delay_time)
      .duration(tran_time)
      .attr("x", new_x)
      .attr("y", new_y)
      .on("end", function () {
        d3.select(this)
          .remove();
      })
    d3.select(d).select("text")
      .clone(true)
      .style("opacity", 0)
      .transition()
      .delay(delay_time)
      .duration(tran_time)
      .style("opacity", 1)
      .attr("x", new_x + new_wd / 2)
      .attr("y", new_y + height / 2)
      .on("end", function () {
        insertnodeinto(d3.select(this).node(), ".r_tbl");
      })
  })
  delay_time = delay_time + tran_time;
  return delay_time;
}
function gather_anim_key(tbl_mid_xy, current_chunk, col,
    col_ind, key, key_ind, speed = 1, start_time = 0,
        msg = false) {
  let delay_time = start_time;
  let tran_time = 2000 / speed;
  let tran_time2 = 3000 / speed;
  let iso_time = 4000 / speed;
  let pause_time = 1500 / speed;
  let height = parseFloat(d3.select("rect").attr("height"));
  let o_key_header = d3.select(".o_rows")
    .select(`.o_cols:nth-child(${col_ind})`).node();
  let r_key_els = d3.selectAll(current_chunk)
    .select(`.r_cols:nth-child(${key_ind})`).nodes();
  if (msg === true) {
    iso_msgbox(tbl_mid_xy["x"], tbl_mid_xy["y"],
        height * 10, height, `Move accross ${col} under ${key}`,
      delay_time, iso_time, 0.5, pause_time);
    delay_time = delay_time + iso_time;
  }
  o_key_header_rect = d3.select(o_key_header).select("rect")
```

```
    .node();
  o_key_header_text = d3.select(o_key_header).select("text")
    .node();
  base_xy = rect_mid_cord(o_key_header_rect)[0];
  db_pulse_rect(o_key_header_rect, "yellow", tran_time,
    delay_time);
  r_key_els.forEach(function (d, i) {
    new_rect = d3.select(d).select("rect").node();
    new_xy = rect_mid_cord(new_rect)[0];
    db_pulse_rect(new_rect, "yellow", tran_time, delay_time);
    animate_line(base_xy["x"], new_xy["x"], base_xy["y"],
        new_xy["y"], tran_time, delay_time);
  })
  delay_time = delay_time + tran_time;
  r_key_els.forEach(function (d, i) {
    new_rect = d3.select(d).select("rect");
    new_x = parseFloat(new_rect.attr("x"));
    new_y = parseFloat(new_rect.attr("y"));
    new_wd = parseFloat(new_rect.attr("width"));
    d3.select(o_key_header_rect)
      .clone(true)
      .style("opacity", 0)
      .transition()
      .delay(delay_time)
      .duration(tran_time)
      .style("opacity", 1)
      .attr("x", new_x)
      .attr("y", new_y)
      .attr("width", new_wd)
      .on("end", function () {
        d3.select(this)
          .remove();
      })
    d3.select(o_key_header_text)
      .clone(true)
      .style("opacity", 0)
      .transition()
      .delay(delay_time)
      .duration(tran_time)
      .style("opacity", 1)
      .attr("x", new_x + new_wd / 2)
      .attr("y", new_y + height / 2)
      .on("end", function () {
        insertnodeinto(d3.select(this).node(), ".r_tbl")
      })
  })
  delay_time = delay_time + tran_time;
  return delay_time;
}
function gather_anim_value(tbl_mid_xy, current_chunk, col,
    col_ind, value, value_ind, speed = 1, start_time = 0,
```

```
      msg = false) {
d3.selectAll(".r_rows").style("opacity", 1)
let delay_time = start_time;
let tran_time = 2000 / speed;
let tran_time2 = 3000 / speed;
let iso_time = 4000 / speed;
let pause_time = 1500 / speed;
let height = parseFloat(d3.select("rect").attr("height"));
let o_value_els = d3.selectAll(".o_rows").selectAll
  (`.o_cols:nth-child(${col_ind})`).nodes();
o_value_els.shift();
let r_value_els = d3.selectAll(current_chunk)
  .select(`.r_cols:nth-child(${value_ind})`).nodes();
if (msg === true) {
  iso_msgbox(tbl_mid_xy["x"], tbl_mid_xy["y"], height * 15,
  height, `Move the values under
      ${col} into the ${value} column`,
    delay_time, iso_time, 0.5, pause_time);
  delay_time = delay_time + iso_time;
}
r_value_els.forEach(function (d, i) {
  base_rect = d3.select(o_value_els[i]).select('rect');
  base_text = d3.select(o_value_els[i]).select('text');
  new_rect = d3.select(d).select("rect");
  new_x = parseFloat(new_rect.attr("x"));
  new_y = parseFloat(new_rect.attr("y"));
  new_wd = parseFloat(new_rect.attr("width"));
  base_rect.clone(true)
    .style("opacity", 0)
    .transition()
    .delay(delay_time)
    .duration(tran_time)
    .style("opacity", 1)
    .attr("x", new_x)
    .attr("y", new_y)
    .attr("width", new_wd)
    .on("end", function () {
      d3.select(this)
        .remove();
    });
  base_text.clone(true)
    .style("opacity", 0)
    .transition()
    .delay(delay_time)
    .duration(tran_time)
    .style("opacity", 1)
    .attr("x", new_x + new_wd / 2)
    .attr("y", new_y + height / 2)
    .on("end", function () {
      insertnodeinto(d3.select(this).node(), ".r_tbl")
    })
```

```
  })
  delay_time = delay_time + tran_time;
  return delay_time;
}

function prepare_spread(start_xy, key, value, key_ind,
    key_width, value_width, key_seq, height,
        start_time = 0, speed = 1) {
  let tran_time = 2000 / speed;
  let delay_time = start_time;
  anim_header = d3.select("svg")
    .append("g")
    .attr("id", "anim_header");
  anim_header
    .append("text")
    .attr("id", "key_txt1")
    .attr("x", start_xy["x"])
    .attr("y", start_xy["y"])
    .style("opacity", 0)
    .style("text-anchor", "start")
    .style("font-size", height * 0.6)
    .text("Use column ");
  key_rect_x = d3.select("#key_txt1").node()
    .getComputedTextLength() * 1.025 +
    start_xy["x"];
  key_rect_y = start_xy["y"] - height / 2;
  anim_header.append("g")
    .attr("id", "key_el")
    .append("rect")
    .style("opacity", 0)
    .attr("id", "key_rect")
    .attr("x", key_rect_x)
    .attr("y", key_rect_y)
    .attr("width", key_width)
    .attr("height", height);
  d3.select("#key_el")
    .append("text")
    .attr("id", "key_text")
    .attr("x", key_rect_x + key_width / 2)
    .attr("y", key_rect_y + height / 2)
    .text(key)
    .style("opacity", 0)
    .style("font-size", height * 0.5);
  anim_header.select("#key_txt1")
    .append("tspan")
    .attr("id", "key_txt2")
    .attr("x", key_rect_x + key_width * 1.05)
    .text(" to spread out to columns")
  d3.select("#key_txt1")
    .transition()
    .delay(start_time)
```

```
    .duration(tran_time)
    .style("opacity", 1);
  d3.select("#key_rect")
    .transition()
    .delay(start_time)
    .duration(tran_time)
    .style("opacity", 1);
  d3.select("#key_text")
    .transition()
    .delay(start_time)
    .duration(tran_time)
    .style("opacity", 1);
  delay_time = delay_time + tran_time;
  keyel_rect_x = d3.select("#key_txt1").node()
    .getComputedTextLength() + key_width * 1.2;
  adj_width = 0;
  adj_text_x = 0;
  keyel_nodes = d3.selectAll(".o_rows")
    .selectAll(`.o_cols:nth-child(${key_ind})`)
    .nodes();
  for (let i = 0; i < key_seq.length; i++) {
    cur_head_move = d3.select(keyel_nodes[key_seq[i]["start"]]);
    cur_move_rect = cur_head_move.select("rect").clone(true);
    cur_move_text = cur_head_move.select("text").clone(true);
    insertnodeinto(cur_move_rect.node(), "#anim_header");
    insertnodeinto(cur_move_text.node(), "#anim_header");
    cur_move_rect
      .transition()
      .delay(delay_time)
      .duration(tran_time)
      .attr("x", keyel_rect_x + adj_width)
      .attr("y", key_rect_y);
    cur_move_text
      .transition()
      .delay(delay_time)
      .duration(tran_time)
      .attr("x", keyel_rect_x + key_width / 2 + adj_width)
      .attr("y", key_rect_y + height / 2);
    adj_width = adj_width + parseFloat(cur_move_rect
        .attr("width")) * 1.05;
  }
  delay_time = delay_time + tran_time;
  start_xy["y"] = start_xy["y"] + height * 1.2;
  anim_header
    .append("text")
    .attr("id", "value_txt")
    .attr("x", start_xy["x"])
    .attr("y", start_xy["y"])
    .style("opacity", 0)
    .style("text-anchor", "start")
    .style("font-size", height * 0.6)
```

```
    .text("Then, put their corresponding ");
  value_rect_x = d3.select("#value_txt").node()
    .getComputedTextLength() * 1.015 +
    start_xy["x"];
  value_rect_y = start_xy["y"] - height / 2;
  anim_header.append("g")
    .attr("id", "value_el")
    .append("rect")
    .style("opacity", 0)
    .attr("id", "value_rect")
    .attr("x", value_rect_x)
    .attr("y", value_rect_y)
    .attr("width", value_width)
    .attr("height", height);
  d3.select("#value_el")
    .append("text")
    .style("opacity", 0)
    .attr("id", "value_text")
    .attr("x", value_rect_x + value_width / 2)
    .attr("y", value_rect_y + height / 2)
    .text(value)
    .style("font-size", height * 0.5);
  anim_header.select("#value_txt")
    .append("tspan")
    .attr("id", "key_txt2")
    .attr("x", value_rect_x + value_width * 1.05)
    .text(" values in these columns")
  d3.select("#value_txt")
    .transition()
    .delay(delay_time)
    .duration(tran_time)
    .style("opacity", 1);
  d3.select("#value_rect")
    .transition()
    .delay(delay_time)
    .duration(tran_time)
    .style("opacity", 1);
  d3.select("#value_text")
    .transition()
    .delay(delay_time)
    .duration(tran_time)
    .style("opacity", 1);
  delay_time = delay_time + tran_time;
  return d3.select("svg")
    .transition().delay(delay_time);
}
function spread_anim_pivot(tbl_mid_xy, pivot_name, pivot_rows,
    speed = 1, start_time = 0, msg = false) {
  let delay_time = start_time;
  let tran_time = 2000 / speed;
  let tran_time2 = 3000 / speed;
```

```
  let iso_time = 4000 / speed;
  let pause_time = 1500 / speed;
  let height = parseFloat(d3.select("rect").attr("height"));
  let store = new Array();
  iso_msgbox(tbl_mid_xy["x"], tbl_mid_xy["y"], height * 15,
    height,`Move across unique values from ${pivot_name}`,
        delay_time, iso_time, 0.5, pause_time);
  delay_time = delay_time + iso_time;
  new_piv_nodes = d3.select(".r_tbl").selectAll(".piv_col")
    .selectAll("rect").nodes();
  new_piv_nodes.shift();
  new_piv_width = parseFloat(d3.select(new_piv_nodes[0])
    .attr("width"));
  pivot_rows.forEach(function (d, i) {
    cur_el = d3.selectAll(`.o_rows:nth-child(${d + 1})`)
      .select(`.piv_col`).clone(true);
    cur_el.attr("class", "removed");
    db_pulse_rect(cur_el.select("rect").node(), "yellow",
        tran_time, delay_time);
    cur_x = parseFloat(d3.select(new_piv_nodes[i]).attr("x"));
    cur_y = parseFloat(d3.select(new_piv_nodes[i]).attr("y"));
    cur_el.select("rect")
      .transition()
      .delay(delay_time + tran_time)
      .duration(tran_time)
      .attr("x", cur_x)
      .attr("y", cur_y)
      .on("end", function () {
        d3.select(this)
          .remove()
      });
    cur_el.select("text")
      .transition()
      .delay(delay_time + tran_time)
      .duration(tran_time)
      .attr("x", cur_x + new_piv_width / 2)
      .attr("y", cur_y + height / 2)
      .on("end", function () {
        insertnodeinto(d3.select(this).node(), ".r_tbl");
      });
  })
  delay_time = delay_time + 2 * tran_time;
  return d3.select("svg")
    .transition().delay(delay_time);
}
function spread_anim_move(tbl_mid_xy, chunk, row_seq,
    key_ind, new_key_ind, val_ind, speed = 1,
        start_time = 0, msg = false) {
  let delay_time = start_time;
  let tran_time = 2000 / speed;
  let tran_time2 = 3000 / speed;
```

```
let iso_time = 4000 / speed;
let pause_time = 1500 / speed;
let height = parseFloat(d3.select("rect").attr("height"));
let all_oval_col = d3.selectAll(chunk)
  .select(`.o_cols:nth-child(${val_ind})`).nodes();
let all_r_rows = d3.selectAll(".r_rows").nodes();
let cur_newval_rects = d3.selectAll(all_r_rows)
  .select(`.r_cols:nth-child(${new_key_ind})`)
  .selectAll("rect").nodes()
row_seq.forEach(function (d, i) {
  db_pulse_rect(d3.select(chunk[i]).select(".piv_col")
      .select("rect").node(),
    "yellow", tran_time, delay_time);
  db_pulse_rect(d3.select(chunk[i])
      .select(`.o_cols:nth-child(${key_ind})`)
      .select("rect").node(), "yellow",
          tran_time, delay_time);
  db_pulse_rect(d3.select(".r_rows")
      .select(`.r_cols:nth-child(${new_key_ind})`)
          .select("rect").node(),
    "yellow", tran_time, delay_time);
  db_pulse_rect(d3.select(all_r_rows[d])
      .select(`.r_cols:nth-child(1)`).select("rect").node(),
    "yellow", tran_time, delay_time);
  delay_time = delay_time + tran_time;
  cur_el = d3.select(chunk[i])
      .selectAll(`.o_cols:nth-child(${val_ind})`).clone(true);
  cur_target = d3.select(cur_newval_rects[d]);
  new_x = parseFloat(cur_target.attr("x"));
  new_y = parseFloat(cur_target.attr("y"));
  new_width = parseFloat(cur_target.attr("width"));
  cur_el.select("rect")
    .transition()
    .duration(tran_time)
    .delay(delay_time)
    .attr("x", new_x)
    .attr("y", new_y)
    .attr("width", new_width)
    .on("end", function () {
      d3.select(this)
        .remove();
    })
  cur_el.select("text")
    .transition()
    .duration(tran_time)
    .delay(delay_time)
    .attr("x", new_x + new_width / 2)
    .attr("y", new_y + height / 2)
    .attr("width", new_width)
    .on("end", function () {
      insertnodeinto(d3.select(this).node(), ".r_tbl");
```

```
      });
    delay_time = delay_time + tran_time;
  })
  return delay_time;
}
function spread_anim_fillna(na_pos, speed = 1, start_time = 0) {
  let delay_time = start_time;
  let tran_time = 3000 / speed;
  let height = parseFloat(d3.select("rect").attr("height"));
    na_pos.forEach(function(d, i) {
    ind_x = d[0];
    ind_y = d[1];
  cur_rect = d3.select(`.r_rows:nth-child(${ind_x + 1})`)
      .select(`.r_cols:nth-child(${ind_y})`)
      .select("rect");
    new_x = parseFloat(cur_rect.attr("x"));
    new_y = parseFloat(cur_rect.attr("y"));
    new_width = parseFloat(cur_rect.attr("width"));
    new_na = d3.select("svg")
      .append("text");
    new_na.text("NA")
      .style("font-size", height * 0.5)
      .style("opacity", 0)
      .attr("x", new_x + new_width / 2)
      .attr("y", new_y + height / 2);
    new_na.transition()
      .delay(delay_time)
      .duration(tran_time)
      .style("opacity", 1);
      })
return delay_time;
}
```

Appendix C

shiny app

C.1 Joining module

C.1.1 UI

```
library(shiny)
library(dataAnim)
library(shinyjs)
library(shinyalert)
library(V8)
shinyUI(fluidPage(
  useShinyalert(),
  useShinyjs(),
  extendShinyjs(text =
                  "shinyjs.clearpage = function(){
                    d3.select('svg').remove()}"),
  extendShinyjs(text =
                  "shinyjs.begin = function(){if(d3
                    .select('svg').node() === null)
                            {d3.select('#mypanel')
                                .append('svg')}}"),
  titlePanel("Joining Animation"),
  sidebarLayout(
    sidebarPanel(
      fileInput("xtbl_upload", "Choose CSV File for Table 1",
                multiple = FALSE,
                accept = c("text/csv",
                           "text/comma-separated-values,
                            text/plain", ".csv")),
      fileInput("ytbl_upload", "Choose CSV File for Table 2",
                multiple = FALSE,
                accept = c("text/csv",
                           "text/comma-separated-values,
                            text/plain", ".csv")),
      selectInput("jointype_input", "Join Type", c("Left",
        "Inner", "Complete"),
                  selected = "Left", multiple = FALSE),
      selectInput("join_var", "Variable to join by", NULL,
        multiple = FALSE), sliderInput("speed_sld",
                  "Animation Speed",
                  min = 1,
```

```
                  max = 5,
                  value = 1),
      checkboxInput("msg_chk", "Show annotations", value = F),
      fluidRow(actionButton(inputId = "go_btn", label = "Go!"),
               actionButton(inputId = "clear_btn",
                label = "Clear")), br(),
      fluidRow(column(8), downloadButton(
        "download_btn", "Download Animation")),
      width = 3
    ),
    mainPanel(
      join_animOutput("animpanel0")
    )
  )
))
```

C.1.2 Server

```
library(shiny)
get_id = function(txt, cnt){
  txt = gsub("[0-9]", "", txt)
  return(paste0(txt, cnt))
}
shinyServer(function(input, output, session) {
x_tbl = reactiveValues(data = NULL)
  y_tbl = reactiveValues(data = NULL)
  store = reactiveValues(container_id = "animpanel0",
    cnt = 1, anim = NULL)
  observeEvent(input$go_btn, {
    shinyjs::disable("go_btn")
    output$animpanel0<- join_animRender({
      if(is.null(x_tbl$data) | is.null(y_tbl$data)) {
        return()
      }
      store$anim = dataAnim::join_anim(join_type =
        tolower(input$jointype_input),
            speed = input$speed_sld, x = isolate(x_tbl$data),
                 y = isolate(y_tbl$data), by = input$join_var,
                    show_msg = input$msg_chk)
      return(store$anim)
    })
  })
  observeEvent(c(input$xtbl_upload, input$ytbl_upload), {
    in_x = input$xtbl_upload
    in_y = input$ytbl_upload
    if(is.null(in_x) | is.null(in_y)) {
      return(NULL)
    }
    x_tbl$data = read.csv(in_x$datapath, header = T,
                          stringsAsFactors = F)
    y_tbl$data = read.csv(in_y$datapath, header = T,
```

```
                          stringsAsFactors = F)
    updateSelectInput(session, inputId = "join_var",
                      choices = intersect(colnames(x_tbl$data),
                                          colnames(y_tbl$data)))
  })
  observeEvent(input$clear_btn, {
    js$clear_page()
    shinyjs::enable("go_btn")
  })
  output$download_btn = downloadHandler(
    filename = function() {
      paste0("animation.html")
    },
    content = function(file) {
      if(is.null(store$anim)) {
        shinyalert::shinyalert("Download Failed",
            "Please check if your animation is loaded.",
                type = "error")
        return()
      }
      shinyalert::shinyalert("Download Success",
        "Go check out your animation!", type = "success")
      htmlwidgets::saveWidget(store$anim, file =
        "animation.html", selfcontained = TRUE)
      file.copy('animation.html', file)
    }
  )
})
```

C.2 Reshaping module

C.2.1 UI

```
library(shiny)
library(shinyjs)
library(dataAnim)
shinyUI(fluidPage(
  sidebarLayout(
    sidebarPanel(
      tabsetPanel(

        tabPanel("Data",
                 br(),
                 fileInput("data_upload", "Upload CSV File",
                           multiple = FALSE,
                           accept = c("text/csv",
                             "text/comma-separated-values",
                                ".csv"))
        ),
        tabPanel("Spread",
                 br(),
```

```
                selectInput("s_key", "Select the column to
                spread out to multiple columns (key)", ""),
                selectInput("s_value", "Select the column
                with the values to be put in these
                columns (value)", ""),
                fluidRow(actionButton(inputId = "s_go_btn",
                label = "Go!"),
                    actionButton(inputId = "s_clear_btn",
                        label = "Clear")), br(),
                 fluidRow(column(8), downloadButton(
                    "g_download_btn", "Download Animation"))),
        tabPanel("Gather",
                 br(),
                 textInput("g_key", "Name the new column
                    containing the old column names (Key)"),
                 textInput("g_value", "Name the new column
                    containing the old column values (Value)"),
                 selectInput("g_col",
                    "Select columns to gather together", ""),
                 fluidRow(actionButton(inputId = "g_go_btn",
                    label = "Go!"),
                          actionButton(inputId = "g_clear_btn",
                            label = "Clear")),
                 br(),
                 fluidRow(column(8), downloadButton(
                    "g_download_btn",
                    "Download Animation")))
      )
    ),
    mainPanel(
      spread_animOutput("spread_panel"),
      gather_animOutput("gather_panel")
    )
  )
))
```

C.2.2 Server

```
library(shiny)
library(shinyjs)
library(dataAnim)
shinyServer(function(input, output) {
  observeEvent(input$data_upload, {
    in_x = input$data_upload
    x_tbl$data = read.csv(in_x$datapath, header = T,
                          stringsAsFactorsrshiny = F)
    updateSelectInput(session, inputId = "g_col", choices =
        colnames(x_tbl))
    updateSelectInput(session, inputId = "s_key", choices =
        colnames(x_tbl))
    updateSelectInput(session, inputId = "s_value", choices =
```

```
        colnames(x_tbl))
  })
  observeEvent(input$g_clear_btn, {
    # js$clearpage()
  })
  observeEvent(input$s_clear_btn, {
    # js$clearpage()
  })
  dl = function(file) {
    browser()
    if(is.null(store$anim)) {
      shinyalert::shinyalert("Download Failed",
        "Please check if your animation is loaded.",
            type = "error")
      return()
    }
    shinyalert::shinyalert("Download Success",
        "Go check out your animation!", type = "success")
    htmlwidgets::saveWidget(store$anim, file =
        "animation.html", selfcontained = TRUE)
    file.copy('animation.html', file)
  }
  output$g_download_btn = downloadHandler(
    filename = function() {
      paste0("animation.html")
    },
    content = dl
  )
  output$s_download_btn = downloadHandler(
    filename = function() {
      paste0("animation.html")
    },
    content = dl
  )
})
```

Bibliography

Aden-Buie, Garrick and Tyler Grant Smith (2018). *tidyexplain: Tidy Animated Verbs*. URL: https://CRAN.R-project.org/package=gganimate.

Chang, Winston et al. (2019). *shiny: Web Application Framework for R*. R package version 1.3.2. URL: https://CRAN.R-project.org/package=shiny.

Johnson, Theodore and Tamraparni Dasu (2003). "Data Quality and Data Cleaning: An Overview". In: *SIGMOD Conference*.

Pedersen, Thomas Lin and David Robinson (2019). *gganimate: A Grammar of Animated Graphics*. R package version 1.0.3. URL: https://CRAN.R-project.org/package=gganimate.

Vaidyanathan, Ramnath et al. (2018). *htmlwidgets: HTML Widgets for R*. R package version 1.3. URL: https://CRAN.R-project.org/package=htmlwidgets.

Wickham, Hadley and Garrett Grolemund (2017). *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. 1st. O'Reilly Media, Inc. ISBN: 1491910399, 9781491910399.

Wickham, Hadley and Lionel Henry (2019). *tidyr: Easily Tidy Data with 'spread()' and 'gather()' Functions*. R package version 0.8.3. URL: https://CRAN.R-project.org/package=tidyr.

Wickham, Hadley et al. (2019). *dplyr: A Grammar of Data Manipulation*. R package version 0.8.0.1. URL: https://CRAN.R-project.org/package=dplyr.