Integrating Data from Separate Tables

We have a sample of men and women collected via Prolific, who completed a survey on Qualtrics. The initial sample of participants were recruited via Prolific (“Partner As”). At the end of this survey, participants were asked to enter the email address of their current romantic partner. If “Partner A” opted to enter a valid email address, the survey link was then sent to their romantic partner (“Partner B”).

These data offer the unique opportunity to compare survey responses between both members of a romantic couple. For example, we could compare men’s self-reported experience with erectile dysfunction with their female partner’s perceptions of the male partner’s experience with erectile dysfunction. In other words, these data afford us the opportunity to determine whether both members of a romantic couple generally agree with each other regarding various relationship experiences. However, before we can analyze the data in this way, we’ll need to do some reorganization so that each male partner is paired up with his female counterpart (i.e., each male participant should be in the same row as his female partner). Luckily, all “Partner A” participants were given a 5-digit random ID, which they were asked to provide to their romantic partner (Partner B). Partner B was then asked to provide this 5-digit ID when taking their survey so that their responses could be linked with their partner’s. This 5-digit ID will serve as our “primary key” when joining the responses of men and women into a single table.

When we download the data from Qualtrics, the first thing we notice is that the data are separated into two CSV files: “Partner A Data” and “Partner B Data”, representing those participants who were recruited directly via Prolific (Partner As) and those participants who were recruited by their romantic partner (Partner Bs). Each CSV file contains data from both men and women.

The first thing we want to do after we’ve imported both CSV files into our SQL database is separate each file into two tables by sex (for a total of 4 tables).

create table partner\_a\_men as

select \* from partner\_a\_demographics

where a\_sex\_self = 1;

create table partner\_a\_women as

select \* from partner\_a\_demographics

where a\_sex\_self = 2;

create table partner\_b\_men as

select \* from partner\_b\_demographics

where b\_sex\_self = 1;

create table partner\_b\_women as

select \* from partner\_b\_demographics

where b\_sex\_self = 2;

We can now join the “Partner A” men with their counterparts in the “partner\_b\_women” table, and the “Partner B” men with their counterparts in the “partner\_a\_women” table. We’ll use inner joins so that all columns from each table are preserved when we create our new, combined tables. We’ll also make sure to keep all men on the “left” side of each combined table. This will make it easier to combine all participants into a single table later on.

create table a\_men\_b\_women as

select \* from partner\_a\_men

inner join partner\_b\_women

on partner\_a\_men.a\_random\_id = partner\_b\_women.b\_pa\_Rand\_ID;

create table b\_men\_a\_women as

select \* from partner\_b\_men

inner join partner\_a\_women

on partner\_b\_men.b\_pa\_Rand\_ID = partner\_a\_women.a\_random\_id;

Finally, we need to integrate both tables so that all paired couples are in a single table. But first, we need to rename the columns in both tables so that each column for men’s variables (regardless of whether they were originally partner A or partner B) are the same across tables. And of course, we will repeat this process for women’s variables as well.

I used chatgpt to help me write a couple stored procedures that will automatically rename the columns. For the table in which all men are “partner A” and all women are “partner B”, we want to rename the columns so that the “A” preceding each variable name is replaced with “M” (for men) and each “B” is replaced with “W” for women. Below is an example of just one of the stored procedures which will change all “A”s to “M”s, but we only need to change a few bits of code to make all of the necessary changes.

DELIMITER //

CREATE PROCEDURE RenameColumns()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE oldColumnName VARCHAR(255);

DECLARE newColumnName VARCHAR(255);

DECLARE columnDataType VARCHAR(255);

DECLARE isNullable VARCHAR(3);

DECLARE columnDefault VARCHAR(255);

DECLARE extraInfo VARCHAR(255);

DECLARE columnCursor CURSOR FOR

SELECT COLUMN\_NAME, DATA\_TYPE, IS\_NULLABLE, COLUMN\_DEFAULT, EXTRA

FROM INFORMATION\_SCHEMA.COLUMNS

WHERE TABLE\_NAME = 'a\_men\_b\_women' AND COLUMN\_NAME LIKE 'A%';

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN columnCursor;

column\_loop: LOOP

FETCH columnCursor INTO oldColumnName, columnDataType, isNullable, columnDefault, extraInfo;

IF done THEN

LEAVE column\_loop;

END IF;

SET newColumnName = CONCAT('M', SUBSTRING(oldColumnName, 2));

SET @sql = CONCAT('ALTER TABLE b\_men\_a\_women CHANGE COLUMN ', oldColumnName, ' ', newColumnName, ' ', columnDataType,

IF(isNullable = 'NO', ' NOT NULL', ''),

IF(columnDefault IS NOT NULL, CONCAT(' DEFAULT ', QUOTE(columnDefault)), ''),

IF(extraInfo IS NOT NULL, CONCAT(' ', extraInfo), ''));

PREPARE stmt FROM @sql;

EXECUTE stmt;

DEALLOCATE PREPARE stmt;

END LOOP;

CLOSE columnCursor;

END //

DELIMITER ;

-- Call the stored procedure

CALL RenameColumns();

Now, we can use the “UNION” operator to vertically stack/concatenate the two tables

create table all\_couples as

select \* from a\_men\_b\_women

union

select \* from b\_men\_a\_women;

Data Cleaning

Now that the data have been integrated into a single table, we can start cleaning (i.e., removing cases that violate our inclusion criteria).