Merging Data from Two Tables

Goal

The Data

Joining the tables

Inner join

Left join

Right join

Full join

Comparison

Exercises

Getting credit

Code ▾

STAT 209: Lab 7

Merging Data from Two Tables

Goal

To become comfortable with the so-called **mutating join** operations: verbs that create one table out of two, where the result table may have more variables (columns) than either component table. (This is in contrast to **filtering joins**, which use information in a second table to extract a subset of cases from the first)

The Data

We'll work (surprise, surprise) with data on babynames, but this time in addition to the Social Security database, we'll also use data from the U.S. Census.

Preliminaries

Load libraries and data

Code:

Code

Let's peek at the `births` table, since we haven't used it before:

Code

```
## # A tibble: 6 x 2
##    year  births
##   <int>   <int>
## 1  1909 2718000
## 2  1910 2777000
## 3  1911 2809000
## 4  1912 2840000
## 5  1913 2869000
## 6  1914 2966000
```

This is just the total number of births in the U.S. for each year, going back to 1909.

Our goal will be to create a single dataset indexed by year that contains one column for the total number of births that year as recorded by the census, and another column that contains the total number of births that year as recorded by the Social Security Administration.

First, let's create a summarized version of the SSA data which is indexed by year, and records the total number of births that year. To make the differences between the join operations clearer, let's also chop off the data after 2012.

Code:

Code

```
## # A tibble: 6 x 3
##    year distinct_name_sex_combos births
##   <dbl>                    <int>  <int>
## 1  1880                     2000 201484
## 2  1881                     1935 192696
## 3  1882                     2127 221533
## 4  1883                     2084 216946
## 5  1884                     2297 243462
## 6  1885                     2294 240854
```

Just for clarity (and parallel naming), let's make a copy of the `births` dataset called `census_births` .

Code:

Code

Joining the tables

Note that both `ssa_births` and `census_births` have a `year` column, with one entry per year. However, they cover different sets of years.

Code

```
## # A tibble: 6 x 3
##    year distinct_name_sex_combos births
##   <dbl>                    <int>  <int>
## 1  1880                     2000 201484
## 2  1881                     1935 192696
## 3  1882                     2127 221533
## 4  1883                     2084 216946
## 5  1884                     2297 243462
## 6  1885                     2294 240854
```

Code

```
## # A tibble: 6 x 2
##    year  births
##   <int>   <int>
## 1  1909 2718000
## 2  1910 2777000
## 3  1911 2809000
## 4  1912 2840000
## 5  1913 2869000
## 6  1914 2966000
```

If we want to combine them into a single table, one decision we have to make is what to do with the years that are in one table but not the other? The way we answer this question determines the type of `join` we will use.

Inner join

The `inner_join()` operation matches entries based on a “key” variable, and retains a case only if the key exists in both the left and the right table.

Compare the years in the `ssa_births` data:

Code

```
##   [1] 1880 1881 1882 1883 1884 1885 1886 1887 1888 1889 1890 1891 1892 1893
##  [15] 1894 1895 1896 1897 1898 1899 1900 1901 1902 1903 1904 1905 1906 1907
##  [29] 1908 1909 1910 1911 1912 1913 1914 1915 1916 1917 1918 1919 1920 1921
##  [43] 1922 1923 1924 1925 1926 1927 1928 1929 1930 1931 1932 1933 1934 1935
##  [57] 1936 1937 1938 1939 1940 1941 1942 1943 1944 1945 1946 1947 1948 1949
##  [71] 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960 1961 1962 1963
##  [85] 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977
##  [99] 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991
## [113] 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005
## [127] 2006 2007 2008 2009 2010 2011 2012
```

to the years in the `census_births` data:

Code

```
##   [1] 1909 1910 1911 1912 1913 1914 1915 1916 1917 1918 1919 1920 1921 1922
##  [15] 1923 1924 1925 1926 1927 1928 1929 1930 1931 1932 1933 1934 1935 1936
##  [29] 1937 1938 1939 1940 1941 1942 1943 1944 1945 1946 1947 1948 1949 1950
##  [43] 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960 1961 1962 1963 1964
##  [57] 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978
##  [71] 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992
##  [85] 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006
##  [99] 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017
```

to the years in the data resulting from an `inner_join()` :

Code

```
## # A tibble: 6 x 4
##    year distinct_name_sex_combos births.x births.y
##   <dbl>                    <int>    <int>    <int>
## 1  1909                     4227   511228  2718000
## 2  1910                     4629   590715  2777000
## 3  1911                     4867   644279  2809000
## 4  1912                     6351   988064  2840000
## 5  1913                     6968  1137111  2869000
## 6  1914                     7965  1416343  2966000
```

Code

```
##   [1] 1909 1910 1911 1912 1913 1914 1915 1916 1917 1918 1919 1920 1921 1922
##  [15] 1923 1924 1925 1926 1927 1928 1929 1930 1931 1932 1933 1934 1935 1936
##  [29] 1937 1938 1939 1940 1941 1942 1943 1944 1945 1946 1947 1948 1949 1950
##  [43] 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960 1961 1962 1963 1964
##  [57] 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978
##  [71] 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992
##  [85] 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006
##  [99] 2007 2008 2009 2010 2011 2012
```

Only the years in *both* tables are included. Note that since both tables had a variable called `births` , but since this was not the key used in the join operation, R created new variable names to disambiguate. The one with the suffix `.x` comes from the left-hand table (`ssa_births` in this case), and the one with the suffix `.y` comes from the right-hand table (`census_births`).

Left join

In contrast, if we use `left_join()` then we will keep every entry from the left table, and record data from the variables added by the right table as `NA` for “missing”.

Code

```
## # A tibble: 39 x 4
##     year distinct_name_sex_combos births.x births.y
##    <dbl>                    <int>    <int>    <int>
##  1  1880                     2000   201484       NA
##  2  1881                     1935   192696       NA
##  3  1882                     2127   221533       NA
##  4  1883                     2084   216946       NA
##  5  1884                     2297   243462       NA
##  6  1885                     2294   240854       NA
##  7  1886                     2392   255317       NA
##  8  1887                     2373   247394       NA
##  9  1888                     2651   299473       NA
## 10  1889                     2590   288946       NA
## # … with 29 more rows
```

Code

```
## # A tibble: 4 x 4
##    year distinct_name_sex_combos births.x births.y
##   <dbl>                    <int>    <int>    <int>
## 1  2009                    34702  3815638  4130665
## 2  2010                    34067  3690700  3999386
## 3  2011                    33903  3651914  3953590
## 4  2012                    33732  3650462  3952841
```

Code

```
##   [1] 1880 1881 1882 1883 1884 1885 1886 1887 1888 1889 1890 1891 1892 1893
##  [15] 1894 1895 1896 1897 1898 1899 1900 1901 1902 1903 1904 1905 1906 1907
##  [29] 1908 1909 1910 1911 1912 1913 1914 1915 1916 1917 1918 1919 1920 1921
##  [43] 1922 1923 1924 1925 1926 1927 1928 1929 1930 1931 1932 1933 1934 1935
##  [57] 1936 1937 1938 1939 1940 1941 1942 1943 1944 1945 1946 1947 1948 1949
##  [71] 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960 1961 1962 1963
##  [85] 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977
##  [99] 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991
## [113] 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005
## [127] 2006 2007 2008 2009 2010 2011 2012
```

Notice that missing values are recorded for years that are included in `ssa_births` but not in `census_births`; but for years included in `census_births` that are not in `ssa_births` (that is, years after 2012), the entries from `census_births` are omitted entirely.

We can examine this explicitly by filtering to see only those rows for which one or the other variable has missing data, using the `is.na()` function:

Code

Code

```
## # A tibble: 0 x 4
## # … with 4 variables: year <dbl>, distinct_name_sex_combos <int>,
## #   births.x <int>, births.y <int>
```

Code

```
## # A tibble: 29 x 4
##     year distinct_name_sex_combos births.x births.y
##    <dbl>                    <int>    <int>    <int>
##  1  1880                     2000   201484       NA
##  2  1881                     1935   192696       NA
##  3  1882                     2127   221533       NA
##  4  1883                     2084   216946       NA
##  5  1884                     2297   243462       NA
##  6  1885                     2294   240854       NA
##  7  1886                     2392   255317       NA
##  8  1887                     2373   247394       NA
##  9  1888                     2651   299473       NA
## 10  1889                     2590   288946       NA
## # … with 19 more rows
```

It seems that there are duplicate entries for the years from 2002 on. This is due to duplicate entries in the original `births` dataset. I'm not sure why this happened since the entries are identical, but as it happens it illustrates an aspect of `join` s: if a key is found multiple times in one dataset, then the data from the other dataset is duplicated.

Right join

Right join (`right_join()`) is just the opposite of left join, with the roles switched. In fact, `df2 %>% right_join(df1)` is the same as `df1 %>% left_join(df2)` .

Full join

If we want to keep entries for years that appear in *either* dataset, we can perform a **full join**. In this case, whichever years are missing in one or the other dataset will have an `NA` for that variable.

Code

```
##   [1] 1880 1881 1882 1883 1884 1885 1886 1887 1888 1889 1890 1891 1892 1893
##  [15] 1894 1895 1896 1897 1898 1899 1900 1901 1902 1903 1904 1905 1906 1907
##  [29] 1908 1909 1910 1911 1912 1913 1914 1915 1916 1917 1918 1919 1920 1921
##  [43] 1922 1923 1924 1925 1926 1927 1928 1929 1930 1931 1932 1933 1934 1935
##  [57] 1936 1937 1938 1939 1940 1941 1942 1943 1944 1945 1946 1947 1948 1949
##  [71] 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960 1961 1962 1963
##  [85] 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977
##  [99] 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991
## [113] 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005
## [127] 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017
```

Code

```
## # A tibble: 5 x 4
##    year distinct_name_sex_combos births.x births.y
##   <dbl>                    <int>    <int>    <int>
## 1  2013                       NA       NA  3932181
## 2  2014                       NA       NA  3988076
## 3  2015                       NA       NA  3978497
## 4  2016                       NA       NA  3945875
## 5  2017                       NA       NA  3855500
```

Code

```
## # A tibble: 29 x 4
##     year distinct_name_sex_combos births.x births.y
##    <dbl>                    <int>    <int>    <int>
##  1  1880                     2000   201484       NA
##  2  1881                     1935   192696       NA
##  3  1882                     2127   221533       NA
##  4  1883                     2084   216946       NA
##  5  1884                     2297   243462       NA
##  6  1885                     2294   240854       NA
##  7  1886                     2392   255317       NA
##  8  1887                     2373   247394       NA
##  9  1888                     2651   299473       NA
## 10  1889                     2590   288946       NA
## # … with 19 more rows
```

Notice that the full set of years is the union of the first two sets.

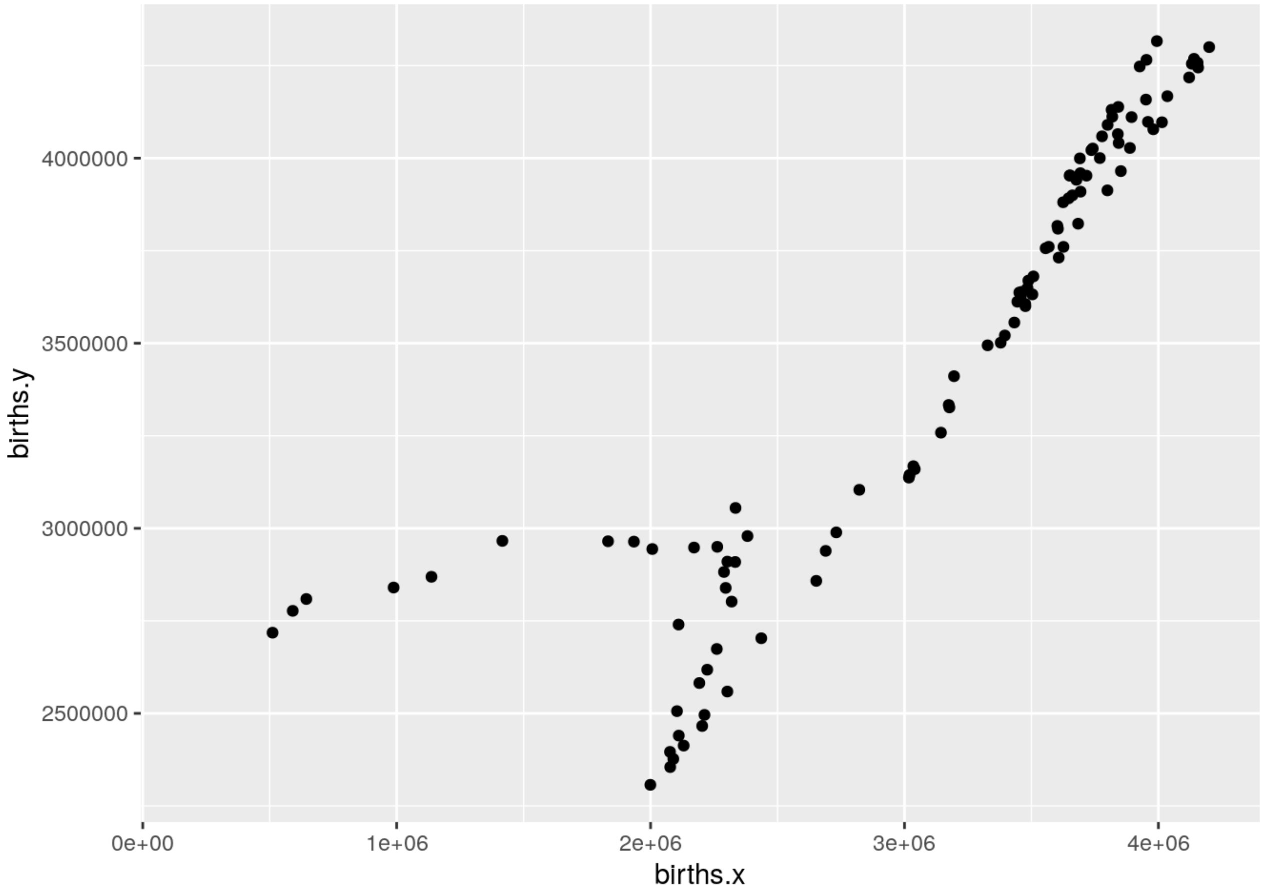
Comparison

Having joined the data from the two sources, let’s see to what extent two sources agree by creating a scatterplot of the births with the data from `ssa_births` on the x axis and the data from `census_births` on the y.

Code

Code

```
## Warning: Removed 34 rows containing missing values (geom_point).
```



Exercise 1 Examine the documentation for the two original datasets to see whether you can account for the discrepancies.

Exercises

The following use the `nycflights13` package.

Exercise 2 Compute the average arrival delay time (`arr_delay`) for each carrier (this requires a `group_by()` and `summarize()`), and include the full name of the carrier in your result set.

Sample solution:

Code

Exercise 3 What was the full name of the airport that was the most common destination from NYC in 2013? How many flights went there that year? (Hint: Group by destination and count the number of flights to each destination with a `summarize()` ; then use a second `summarize()` to return the destination that had the highest number of flights, as well as the actual number of flights to that destination. Finally, join the resulting one-line table with the `airports` table to add on the actual name of the airport. Make sure you are using a type of join that keeps the final result at just one entry. Optional: Restrict the columns in the final output to just the airport code, airport name, and number of flights.)

Sample solution:

Code

Exercise 4 What model of plane had the most total flights in 2013? The frequency data comes from the `flights` table, and the model of each plane comes from the `planes` table. First, join the `flights` table with the `planes` table in a way such that only planes whose tail number corresponds to a known model are included. Then, group the resulting table by model. Then count the number of flights for each model using `summarize()` . With a second `summarize()` , find the model with the highest number of flights, as well as the number of flights itself.

Sample solution:

Code

Exercise 5 Were there any flights that went to "mystery" airports (i.e., airports that don't appear in the airports table)? What were they and how many flights went to each of those airports? Perform a join to get the airport name for each destination in the `flights` table, producing `NA` for any flights whose destination code does not appear in the `airports` table. Filter the results to retain only those flights heading to these "mystery" airports. Then group the data by destination and count the number of flights going to each mystery airport. Sort the final results to show the most popular mystery airports first (i.e., in descending order of number of flights).

Sample solution:

Code

Exercise 6 Were there any "mystery" planes (i.e., planes that don't appear in the planes table)? Produce output in the same form as for the previous exercise, with the tail numbers of the mystery planes, and the number of flights flown by those planes, sorted in descending order based on which planes took the most flights. No hints this time!

Sample solution: REDACTED

Getting credit

DM me on Slack with the Honor Pledge (certifying that you completed the lab, and made a good faith effort to work out the exercises on your own before peeking. Then answer the following prompt (also in a DM): You have now learned the basics of data wrangling. What remains unclear?