# 9.2 Types of Joins

May 9, 2019

## 1 9.2 Types of Joins

In the previous section, we discussed how to *merge* (or *join*) two data sets by matching on certain variables. But what happens when no match can be found for a row in one `DataFrame`?

First, let's determine how *pandas* handles this situation by default. The name "Nevaeh", which is "Heaven" spelled backwards, is said to have taken off when Sonny Sandoval of the band P.O.D. gave his daughter the name in 2000. Let's look at how common this name was four years earlier and four years after.

In [1]:
```
import pandas as pd
pd.options.display.max_rows = 8

names1996 = pd.read_csv("http://github.com/dlsun/data-science-book/blob/"
                        "master/data/names/yob1996.txt?raw=true",
                        header=None,
                        names=["Name", "Sex", "Count"])
names2004 = pd.read_csv("http://github.com/dlsun/data-science-book/blob/"
                        "master/data/names/yob2004.txt?raw=true",
                        header=None,
                        names=["Name", "Sex", "Count"])
```

In [2]:
```
names1996[names1996.Name == "Nevaeh"]
```

Out[2]:
```
Empty DataFrame
Columns: [Name, Sex, Count]
Index: []
```

In [3]:
```
names2004[names2004.Name == "Nevaeh"]
```

Out[3]:
```
         Name Sex  Count
103    Nevaeh   F   3179
21758  Nevaeh   M     35
```

In 1996, there were no girls (or fewer than 5) named Nevaeh; just eight years later, there were over 3000 girls (and 27 boys) with the name. It seems like Sonny Sandoval had a huge effect.

What will happen to the name "Nevaeh" when we merge the two data sets?

In [4]:
```
names = names1996.merge(names2004, on=["Name", "Sex"])
names[names.Name == "Nevaeh"]
```

```
Out[4]: Empty DataFrame
        Columns: [Name, Sex, Count_x, Count_y]
        Index: []
```

By default, *pandas* only includes combinations that are present in *both* `DataFrames`. If it cannot find a match for a row in one `DataFrame`, then the combination is simply dropped.

But in this context, the fact that a name does not appear in one data set is informative. It means that no babies were born in that year with that name. (Technically, it means that fewer than 5 babies were born with that name, as any name that was assigned fewer than 5 times is omitted for privacy reasons.) We might want to include names that appeared in only one of the two `DataFrames`, rather than just the names that appeared in both.

There are four types of joins, distinguished by whether they include names from the left `DataFrame`, the right `DataFrame`, both, or neither:

1. **inner join** (default): only values that are present in *both* `DataFrames` are included in the result
2. **outer join**: any value that appears in *either* `DataFrame` is included in the result
3. **left join**: any value that appears in the *left* `DataFrame` is included in the result, whether or not it appears in the right `DataFrame`
4. **right join**: any value that appears in the *right* `DataFrame` is included in the result, whether or not it appears in the left `DataFrame`.

In *pandas*, the join type is specified using the `how=` argument.

Now let's look at examples of each of these types of joins.

```
In [5]: # inner join
        names_inner = names1996.merge(names2004, on=["Name", "Sex"], how="inner")
        names_inner
```

```
Out[5]:           Name Sex  Count_x  Count_y
        0        Emily   F    25150    25028
        1      Jessica   F    24192     9469
        2      Ashley    F    23676    14370
        3        Sarah   F    21029    12732
        ...        ...  ..      ...      ...
        20049    Zhane   M        5       15
        20050      Zhi   M        5        9
        20051    Zoran   M        5       14
        20052    Zyler   M        5       23

        [20053 rows x 4 columns]
```

```
In [6]: # outer join
        names_outer = names1996.merge(names2004, on=["Name", "Sex"], how="outer")
        names_outer
```

```
Out[6]:           Name Sex  Count_x  Count_y
        0        Emily   F  25150.0  25028.0
        1      Jessica   F  24192.0   9469.0
        2      Ashley    F  23676.0  14370.0
```

```
3          Sarah   F  21029.0  12732.0
...          ...  ..      ...      ...
38403  Zymarion   M      NaN      5.0
38404    Zymeir   M      NaN      5.0
38405    Zyrell   M      NaN      5.0
38406     Zyron   M      NaN      5.0

[38407 rows x 4 columns]
```

Names like "Zyrell" and "Zyron" appeared in the 2004 data but not the 1996 data. For this reason, their count in 1996 is `NaN`. In general, there will be `NaN`s in a `DataFrame` resulting from an outer join. Any time a name appears in one `DataFrame` but not the other, there will be `NaN`s in the columns from the `DataFrame` whose data is missing.

```
In [7]: names_outer.isnull().sum()

Out[7]: Name           0
        Sex            0
        Count_x    11987
        Count_y     6367
        dtype: int64
```

By contrast, there are no `NaN`s when we do an inner join. That is because we restrict to only the (name, sex) pairs that appeared in both `DataFrame`s, so we have counts for both 1996 and 2014.

```
In [8]: names_inner.isnull().sum()

Out[8]: Name       0
        Sex        0
        Count_x    0
        Count_y    0
        dtype: int64
```

Left and right joins preserve data from one `DataFrame` but not the other. For example, if we were trying to calculate the percentage change for each name from 1996 to 2004, we would want to include all of the names that appeared in the 1996 data. If the name did not appear in the 2004 data, then that is informative.

```
In [9]: # left join
        names_left = names1996.merge(names2004, on=["Name", "Sex"], how="left")
        names_left

Out[9]:            Name Sex  Count_x  Count_y
        0         Emily   F    25150  25028.0
        1       Jessica   F    24192   9469.0
        2        Ashley   F    23676  14370.0
        3         Sarah   F    21029  12732.0
        ...         ...  ..      ...      ...
        26416  Zildjian   M        5      NaN
```

```
26417     Zishe   M        5     NaN
26418     Zoran   M        5    14.0
26419     Zyler   M        5    23.0

[26420 rows x 4 columns]
```

The result of a left join has `NaN`s in the column from the right `DataFrame`.

```
In [10]: names_left.isnull().sum()
```

```
Out[10]: Name          0
         Sex           0
         Count_x       0
         Count_y    6367
         dtype: int64
```

The result of a right join, on the other hand, has `NaN`s in the column from the left `DataFrame`.

```
In [11]: # right join
         names_right = names1996.merge(names2004, on=["Name", "Sex"], how="right")
         names_right
```

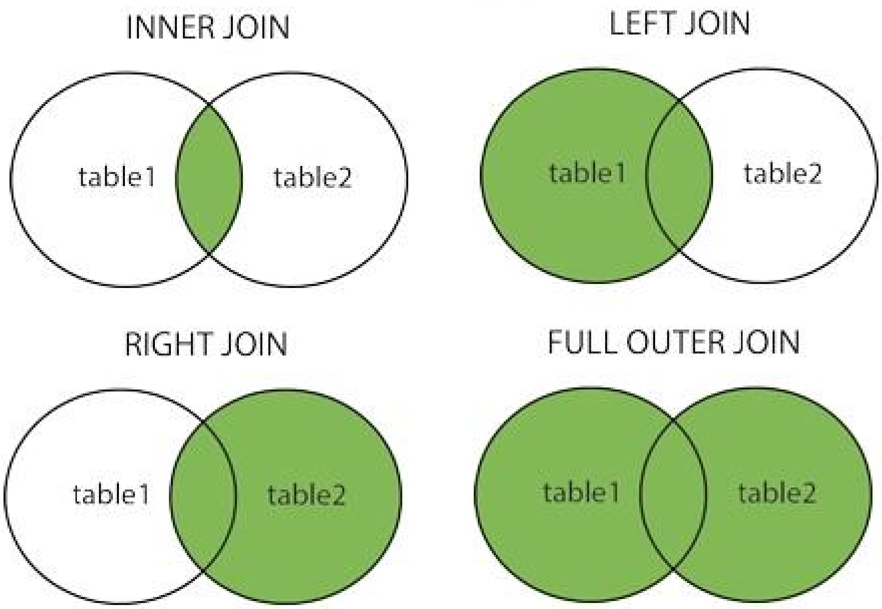
```
Out[11]:            Name Sex  Count_x  Count_y
         0         Emily   F  25150.0    25028
         1       Jessica   F  24192.0     9469
         2        Ashley   F  23676.0    14370
         3         Sarah   F  21029.0    12732
         ...         ...  ..      ...      ...
         32036  Zymarion   M      NaN        5
         32037    Zymeir   M      NaN        5
         32038    Zyrell   M      NaN        5
         32039     Zyron   M      NaN        5

         [32040 rows x 4 columns]
```

```
In [12]: names_right.isnull().sum()
```

```
Out[12]: Name           0
         Sex            0
         Count_x    11987
         Count_y        0
         dtype: int64
```

One way to visualize the different types of joins is using Venn diagrams. The shaded circles specify which values are included in the output.




```
In [ ]: # Exercises
```

```
Exercises 1-2 deal with the Movielens data (`/data301/data/ml-1m/`) that you explored
```

**Exercise 1.** Calculate the number of ratings by movie. How many of the movies had zero ratings?

(*Hint*: Why is an inner join not sufficient here?)

```
In [13]: ratings = pd.read_csv("/data301/data/ml-1m/ratings_small.dat",
                              sep=",",
                              engine="python",
                              header=None,
                              names=["UserID", "MovieID", "Rating", "Timestamp"])
         users = pd.read_csv("/data301/data/ml-1m/users.dat",
                            sep="::",
                            engine="python",
                            header=None,
                            names=["UserID", "Gender", "Age", "Occupation", "Zip-code"])
         movies = pd.read_csv("/data301/data/ml-1m/movies.dat",
                             engine="python",
                             sep="::",
                             header=None,
                             names=["MovieID", "Title", "Genres"])
```

```
In [21]: ratings_movies = ratings.merge(movies, on="MovieID") #inner join means there will be
         ratings_movies.groupby(["Title"])["Rating"].count()
```

```
Out[21]: Title
         'Night Mother (1986)                  2
         'burbs, The (1989)                    2
```

```
...And Justice for All (1979)        2
10 Things I Hate About You (1999)    8
                                    ..
Your Friends and Neighbors (1998)    1
Zed & Two Noughts, A (1985)          1
Zero Effect (1998)                   2
eXistenZ (1999)                      4
Name: Rating, Length: 2291, dtype: int64
```

In [24]:
```
rating_movies = ratings.merge(movies, on="MovieID", how="outer")
rating_movies.groupby(["Title"])["Rating"].count()
```

Out[24]:
```
Title
$1,000,000 Duck (1971)                        0
'Night Mother (1986)                          2
'Til There Was You (1997)                     0
'burbs, The (1989)                            2
                                             ..
Zero Kelvin (Kjrlighetens kjtere) (1995)    0
Zeus and Roxanne (1997)                       0
Zone 39 (1997)                                0
eXistenZ (1999)                               4
Name: Rating, Length: 3883, dtype: int64
```

In [46]:
```
rating_movies[rating_movies.Rating == 0];
rating_movies.Rating = rating_movies.Rating.fillna(0)

rating_movies[rating_movies.Rating == 0]
```

Out[46]:
```
       UserID  MovieID  Rating  Timestamp  \
10000     NaN       20     0.0        NaN
10001     NaN       23     0.0        NaN
10002     NaN       30     0.0        NaN
10003     NaN       31     0.0        NaN
...       ...      ...     ...        ...
11588     NaN     3941     0.0        NaN
11589     NaN     3942     0.0        NaN
11590     NaN     3944     0.0        NaN
11591     NaN     3951     0.0        NaN

                                                   Title        Genres
10000                                 Money Train (1995)        Action
10001                                   Assassins (1995)      Thriller
10002  Shanghai Triad (Yao a yao yao dao waipo qiao) ...         Drama
10003                             Dangerous Minds (1995)         Drama
...                                                  ...           ...
11588                     Sorority House Massacre (1986)        Horror
11589                  Sorority House Massacre II (1990)        Horror
11590                                    Bootmen (2000)  Comedy|Drama
```

```
11591                              Two Family House (2000)         Drama

[1592 rows x 6 columns]
```

**Exercise 2.** How many movies received both a 1 and a 5 rating? Do this by creating and joining two appropriate tables.

```
In [31]: movies_with_1 = ratings_movies[ratings_movies.Rating == 1]
         movies_with_5 = ratings_movies[ratings_movies.Rating == 5]
```

```
In [50]: ones_and_fives = movies_with_1.merge(movies_with_5, on="MovieID")
         ones_and_fives.groupby(["MovieID"])["Title_x"].unique()
```

```
Out[50]: MovieID
         7                                       [Sabrina (1995)]
         11                        [American President, The (1995)]
         25                              [Leaving Las Vegas (1995)]
         94                                [Beautiful Girls (1996)]
                                         ...
         3863                                    [Cell, The (2000)]
         3869    [Naked Gun 2 1/2: The Smell of Fear, The (1991)]
         3949                           [Requiem for a Dream (2000)]
         3952                              [Contender, The (2000)]
         Name: Title_x, Length: 160, dtype: object
```