bonus_merge

February 29, 2024

1 Bonus video: How do I merge DataFrames in pandas?

Full course: pandas in 30 days

1. Selecting a Function
2. Joining (Merging) DataFrames
3. What if…?
4. Four Types of Joins

1.1 Part 1: Selecting a Function

Taken from "Merging DataFrames with pandas" (DataCamp course):

- **`df1.append(df2)`**: stacking vertically
  - *Update from 2024: The "append" method has been removed in favor of the concat function.*
- `pd.concat([df1, df2])`:
  - stacking many horizontally or vertically
  - simple inner/outer joins on Indexes
- `df1.join(df2)`: inner/outer/left/right joins on Indexes
- `pd.merge(df1, df2)`: many joins on multiple columns

1.2 Part 2: Joining (Merging) DataFrames

Using the MovieLens 100k data, let's create two DataFrames:

- **movies**: shows information about movies, namely a unique **movie_id** and its **title**
- **ratings**: shows the **rating** that a particular **user_id** gave to a particular **movie_id** at a particular **timestamp**

[1]:
```
import pandas as pd
```

1.2.1 Movies

[2]:
```
movie_cols = ['movie_id', 'title']
movies = pd.read_table('http://bit.ly/movieitems', sep='|', header=None,␣
  ↪names=movie_cols, usecols=[0, 1])
movies.head()
```

```
[2]:    movie_id               title
     0         1    Toy Story (1995)
     1         2    GoldenEye (1995)
     2         3   Four Rooms (1995)
     3         4   Get Shorty (1995)
     4         5      Copycat (1995)
```

```
[3]: movies.shape
```

```
[3]: (1682, 2)
```

```
[4]: movies.movie_id.nunique()
```

```
[4]: 1682
```

1.2.2 Ratings

```
[5]: rating_cols = ['user_id', 'movie_id', 'rating', 'timestamp']
     ratings = pd.read_table('http://bit.ly/movielensdata', sep='\t', header=None,␣
      ↪names=rating_cols)
     ratings.head()
```

```
[5]:    user_id  movie_id  rating  timestamp
     0      196       242       3  881250949
     1      186       302       3  891717742
     2       22       377       1  878887116
     3      244        51       2  880606923
     4      166       346       1  886397596
```

```
[6]: ratings.shape
```

```
[6]: (100000, 4)
```

```
[7]: ratings.movie_id.nunique()
```

```
[7]: 1682
```

```
[8]: ratings.loc[ratings.movie_id == 1, :].head()
```

```
[8]:       user_id  movie_id  rating  timestamp
     24        308         1       4  887736532
     454       287         1       5  875334088
     957       148         1       4  877019411
     971       280         1       4  891700426
     1324       66         1       3  883601324
```

1.2.3 Merging Movies and Ratings

Let's pretend that you want to examine the ratings DataFrame, but you want to know the **title** of each movie rather than its **movie_id**. The best way to accomplish this objective is by "joining" (or "merging") the DataFrames using the pandas `merge` function:

```
[9]: movies.columns
```

```
[9]: Index(['movie_id', 'title'], dtype='object')
```

```
[10]: ratings.columns
```

```
[10]: Index(['user_id', 'movie_id', 'rating', 'timestamp'], dtype='object')
```

```
[11]: movie_ratings = pd.merge(movies, ratings)
      movie_ratings.columns
```

```
[11]: Index(['movie_id', 'title', 'user_id', 'rating', 'timestamp'], dtype='object')
```

```
[12]: movie_ratings.head()
```

```
[12]:    movie_id             title  user_id  rating  timestamp
      0         1  Toy Story (1995)      308       4  887736532
      1         1  Toy Story (1995)      287       5  875334088
      2         1  Toy Story (1995)      148       4  877019411
      3         1  Toy Story (1995)      280       4  891700426
      4         1  Toy Story (1995)       66       3  883601324
```

```
[13]: movie_ratings.shape
```

```
[13]: (100000, 5)
```

Here's what just happened:

- pandas noticed that movies and ratings had one column in common, namely **movie_id**. This is the "key" on which the DataFrames will be joined.
- The first **movie_id** in movies is 1. Thus, pandas looked through every row in the ratings DataFrame, searching for a movie_id of 1. Every time it found such a row, it recorded the **user_id**, **rating**, and **timestamp** listed in that row. In this case, it found 452 matching rows.
- The second **movie_id** in movies is 2. Again, pandas did a search of ratings and found 131 matching rows.
- This process was repeated for all of the remaining rows in movies.

At the end of the process, the movie_ratings DataFrame is created, which contains the two columns from movies (**movie_id** and **title**) and the three other colums from ratings (**user_id**, **rating**, and **timestamp**).

- **movie_id** 1 and its **title** are listed 452 times, next to the **user_id**, **rating**, and **timestamp** for each of the 452 matching ratings.

- **movie_id** 2 and its **title** are listed 131 times, next to the **user_id**, **rating**, and **timestamp** for each of the 131 matching ratings.
- And so on, for every movie in the dataset.

[14]:
```
print(movies.shape)
print(ratings.shape)
print(movie_ratings.shape)
```

```
(1682, 2)
(100000, 4)
(100000, 5)
```

Notice the shapes of the three DataFrames:

- There are 1682 rows in the movies DataFrame.
- There are 100000 rows in the ratings DataFrame.
- The `merge` function resulted in a movie_ratings DataFrame with 100000 rows, because every row from ratings matched a row from movies.
- The movie_ratings DataFrame has 5 columns, namely the 2 columns from movies, plus the 4 columns from ratings, minus the 1 column in common.

By default, the `merge` function joins the DataFrames using all column names that are in common (**movie_id**, in this case). The documentation explains how you can override this behavior.

1.3 Part 3: What if...?

1.3.1 What if the columns you want to join on don't have the same name?

[15]:
```
movies.columns = ['m_id', 'title']
movies.columns
```

[15]:
```
Index(['m_id', 'title'], dtype='object')
```

[16]:
```
ratings.columns
```

[16]:
```
Index(['user_id', 'movie_id', 'rating', 'timestamp'], dtype='object')
```

[17]:
```
pd.merge(movies, ratings, left_on='m_id', right_on='movie_id').head()
```

[17]:
```
   m_id             title  user_id  movie_id  rating  timestamp
0     1  Toy Story (1995)      308         1       4  887736532
1     1  Toy Story (1995)      287         1       5  875334088
2     1  Toy Story (1995)      148         1       4  877019411
3     1  Toy Story (1995)      280         1       4  891700426
4     1  Toy Story (1995)       66         1       3  883601324
```

1.3.2 What if you want to join on one index?

[18]:
```
movies = movies.set_index('m_id')
movies.head()
```

[18]:
```
                 title
m_id
1     Toy Story (1995)
2     GoldenEye (1995)
3    Four Rooms (1995)
4    Get Shorty (1995)
5       Copycat (1995)
```

[19]:
```
pd.merge(movies, ratings, left_index=True, right_on='movie_id').head()
```

[19]:
```
                 title  user_id  movie_id  rating  timestamp
24    Toy Story (1995)      308         1       4  887736532
454   Toy Story (1995)      287         1       5  875334088
957   Toy Story (1995)      148         1       4  877019411
971   Toy Story (1995)      280         1       4  891700426
1324  Toy Story (1995)       66         1       3  883601324
```

1.3.3 What if you want to join on two indexes?

[20]:
```
ratings = ratings.set_index('movie_id')
ratings.head()
```

[20]:
```
          user_id  rating  timestamp
movie_id
242           196       3  881250949
302           186       3  891717742
377            22       1  878887116
51            244       2  880606923
346           166       1  886397596
```

[21]:
```
pd.merge(movies, ratings, left_index=True, right_index=True).head()
```

[21]:
```
                 title  user_id  rating  timestamp
m_id
1     Toy Story (1995)      308       4  887736532
1     Toy Story (1995)      287       5  875334088
1     Toy Story (1995)      148       4  877019411
1     Toy Story (1995)      280       4  891700426
1     Toy Story (1995)       66       3  883601324
```

1.4 Part 4: Four Types of Joins

There are actually four types of joins supported by the pandas `merge` function. Here's how they are described by the documentation:

- **inner:** use intersection of keys from both frames, similar to a SQL inner join; preserve the order of the left keys
- **outer:** use union of keys from both frames, similar to a SQL full outer join; sort keys lexicographically
- **left:** use only keys from left frame, similar to a SQL left outer join; preserve key order
- **right:** use only keys from right frame, similar to a SQL right outer join; preserve key order

The default is the "inner join", which was used when creating the movie_ratings DataFrame.

It's easiest to understand the different types by looking at some simple examples:

1.4.1 Example DataFrames A and B

[22]:
```
A = pd.DataFrame({'color': ['green', 'yellow', 'red'], 'num':[1, 2, 3]})
A
```

[22]:
```
    color  num
0   green    1
1  yellow    2
2     red    3
```

[23]:
```
B = pd.DataFrame({'color': ['green', 'yellow', 'pink'], 'size':['S', 'M', 'L']})
B
```

[23]:
```
    color size
0   green    S
1  yellow    M
2    pink    L
```

1.4.2 Inner join

Only include observations found in both A and B:

[24]:
```
pd.merge(A, B, how='inner')
```

[24]:
```
    color  num size
0   green    1    S
1  yellow    2    M
```

1.4.3 Outer join

Include observations found in either A or B:

[25]:
```
pd.merge(A, B, how='outer')
```

[25]:
```
    color  num size
0   green  1.0    S
1  yellow  2.0    M
2     red  3.0  NaN
3    pink  NaN    L
```

1.4.4 Left join

Include all observations found in A:

```
[26]: pd.merge(A, B, how='left')
```

```
[26]:     color  num size
      0   green    1    S
      1  yellow    2    M
      2     red    3  NaN
```

1.4.5 Right join

Include all observations found in B:

```
[27]: pd.merge(A, B, how='right')
```

```
[27]:     color  num size
      0   green  1.0    S
      1  yellow  2.0    M
      2    pink  NaN    L
```