# # Joining (Merging) DataFrames

# Using the [MovieLens 100k data](http://grouplens.org/datasets/movielens/), 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\*\*

# ### Movies

import pandas as pd

file\_path = '/Users/jim\_byers/Documents/GA/GA\_Data\_Science\_course/SEA-DAT1/data/'

movie\_url = file\_path + 'u.item'

movie\_cols = ['movie\_id', 'title']

movies = pd.read\_table(movie\_url, sep='|', header=None, names=movie\_cols, usecols=[0, 1])

movies.head()

# ### Ratings

rating\_url = file\_path + 'u.data'

rating\_cols = ['user\_id', 'movie\_id', 'rating', 'timestamp']

ratings = pd.read\_table(rating\_url, sep='\t', header=None, names=rating\_cols)

ratings.head()

# 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:

movie\_ratings = pd.merge(movies, ratings)

movie\_ratings.head()

# 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.

print(movies.shape)

print ratings.shape

print(movie\_ratings.shape)

# 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](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.merge.html) explains how you can override this behavior.

# ## 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 (SQL: inner join)

# - \*\*outer:\*\* use union of keys from both frames (SQL: full outer join)

# - \*\*left:\*\* use only keys from left frame (SQL: left outer join)

# - \*\*right:\*\* use only keys from right frame (SQL: right outer join)

#

# 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:

# ### Example DataFrames A and B

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

A

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

B

# ### Inner join

#

# Only include observations found in both A and B:

pd.merge(A, B, how='inner')

# ### Outer join

#

# Include observations found in either A or B:

pd.merge(A, B, how='outer')

# ### Left join

#

# Include all observations found in A:

pd.merge(A, B, how='left')

# ### Right join

#

# Include all observations found in B:

pd.merge(A, B, how='right')

"""

### Exercise 1: Seattle Pronto Cycle Share data merge

Using the Pronto Cycle Share year one data for https://www.prontocycleshare.com/datachallenge

We want to merge the dockcount values from 2015\_station\_data.csv into 2015\_trip\_data.csv.

In particular we want the dockcount column from 2015\_station\_data.csv to be added to the rows in 2015\_trip\_data.csv.

# to create a single dataframe called trip\_and\_station\_data.

"""

## import pandas as pd

# read in the data from '2015\_trip\_data.csv' file into a dataframe names 'file\_path' and examine the contents.

# name the file trip\_data

file\_path = '/Users/jim\_byers/Documents/GA/GA\_Data\_Science\_course/SEA-DAT1/data/pronto\_cycle\_share/'

trip\_data\_url = file\_path + '2015\_trip\_data.csv'

trip\_data = pd.read\_table(trip\_data\_url, sep=',', header=0)

trip\_data.head()

trip\_data.shape

# read in the data from just columns name and dockcount from 2015\_trip\_data.csv file and examine the contents

# name the file station\_data

station\_data\_url = file\_path + '2015\_station\_data.csv'

station\_data = pd.read\_table(station\_data\_url, sep=',', header=0, usecols=['name','dockcount'])

station\_data.shape

trip\_data.head()

"""

Merge trip\_data with station\_data, joining on trip\_data 'station\_name' and station\_data 'name.

Since we do not know if there is a 'name' in station\_data that matches each station\_name in trip\_data do a left merge

Name the resulting dataframe trip\_and\_dockcount\_data.

"""

trip\_and\_dockcount\_data = pd.merge(trip\_data, station\_data, how='left', left\_on='from\_station\_name', right\_on='name')

trip\_and\_dockcount\_data.shape

trip\_and\_dockcount\_data.columns

# Remove the 'name' column

del trip\_and\_dockcount\_data['name']

trip\_and\_dockcount\_data.shape

trip\_and\_dockcount\_data.columns

# Bonus: check to see if there are any rows that have no value in the dockcount columns.

trip\_and\_dockcount\_data[trip\_and\_dockcount\_data['dockcount'].isnull()]

# data row 74114 has a NaN value in the dockcount column, indicating no value in that column