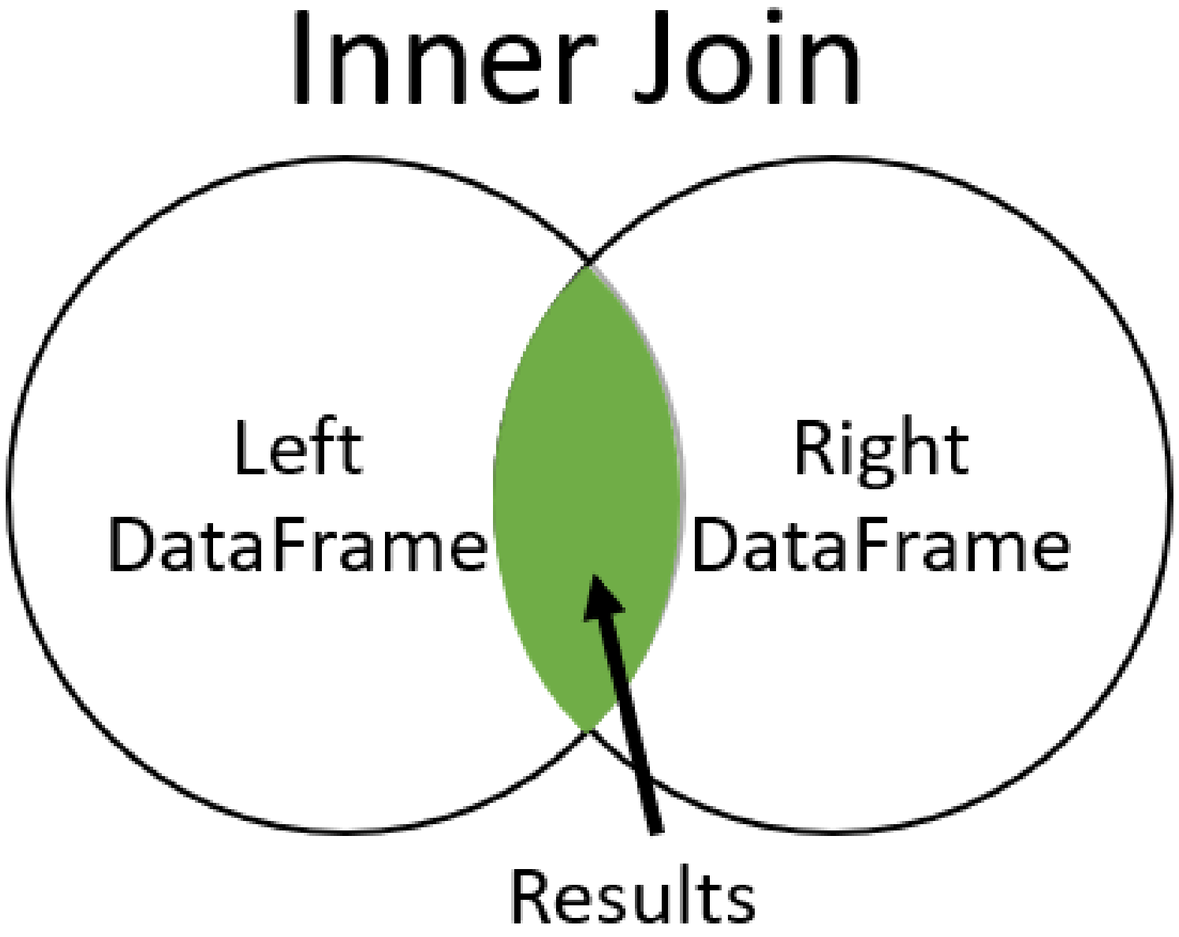
Left join

JOINING DATA WITH PANDAS

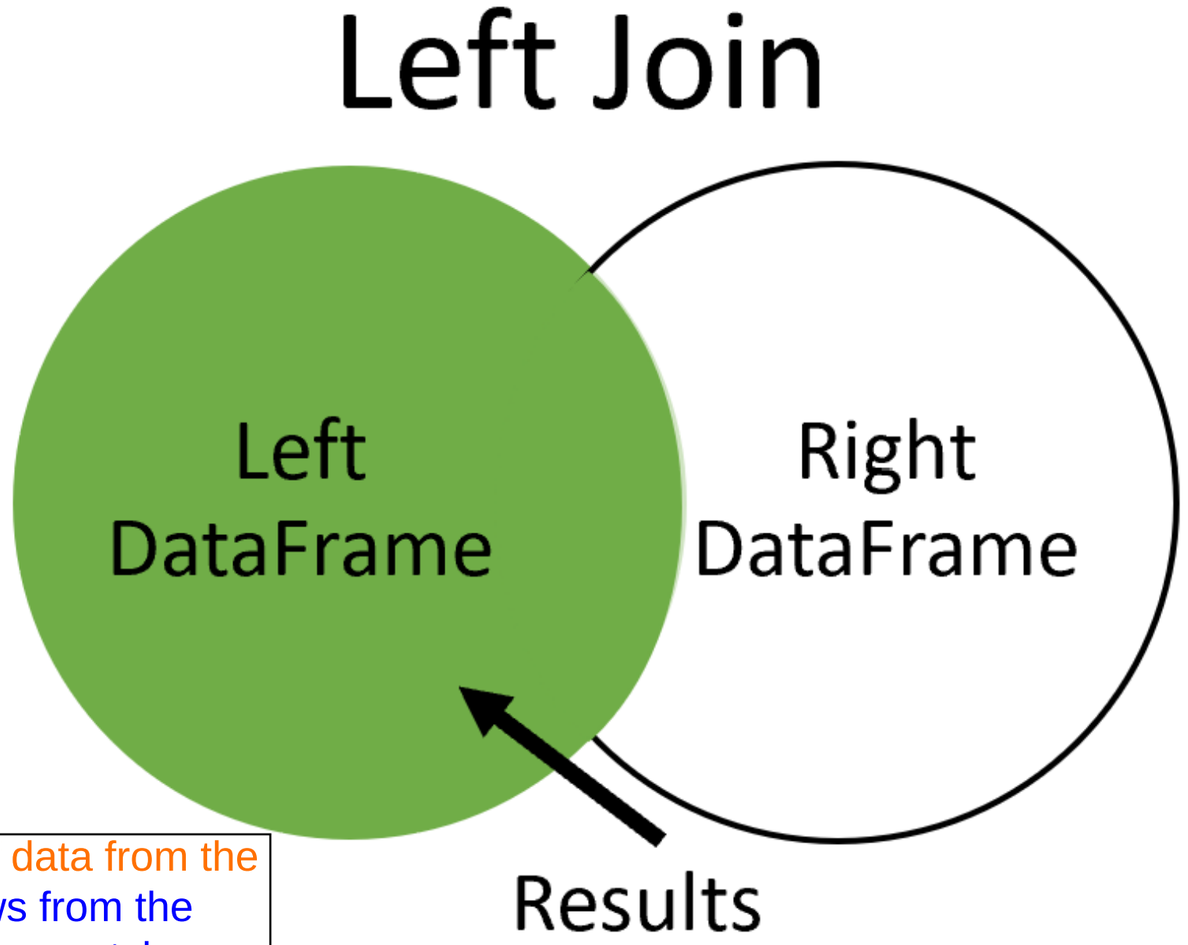
Aaren Stubberfield
Instructor

datacamp

Quick review



Left join



A left join returns all rows of data from the left table and only those rows from the right table where key columns match.

Left join

Left Table

A	B	C
A2	B2	C2
A3	B3	C3
A4	B4	C4

Right Table

C	D
C1	D1
C2	D2
C4	D4
C5	D5

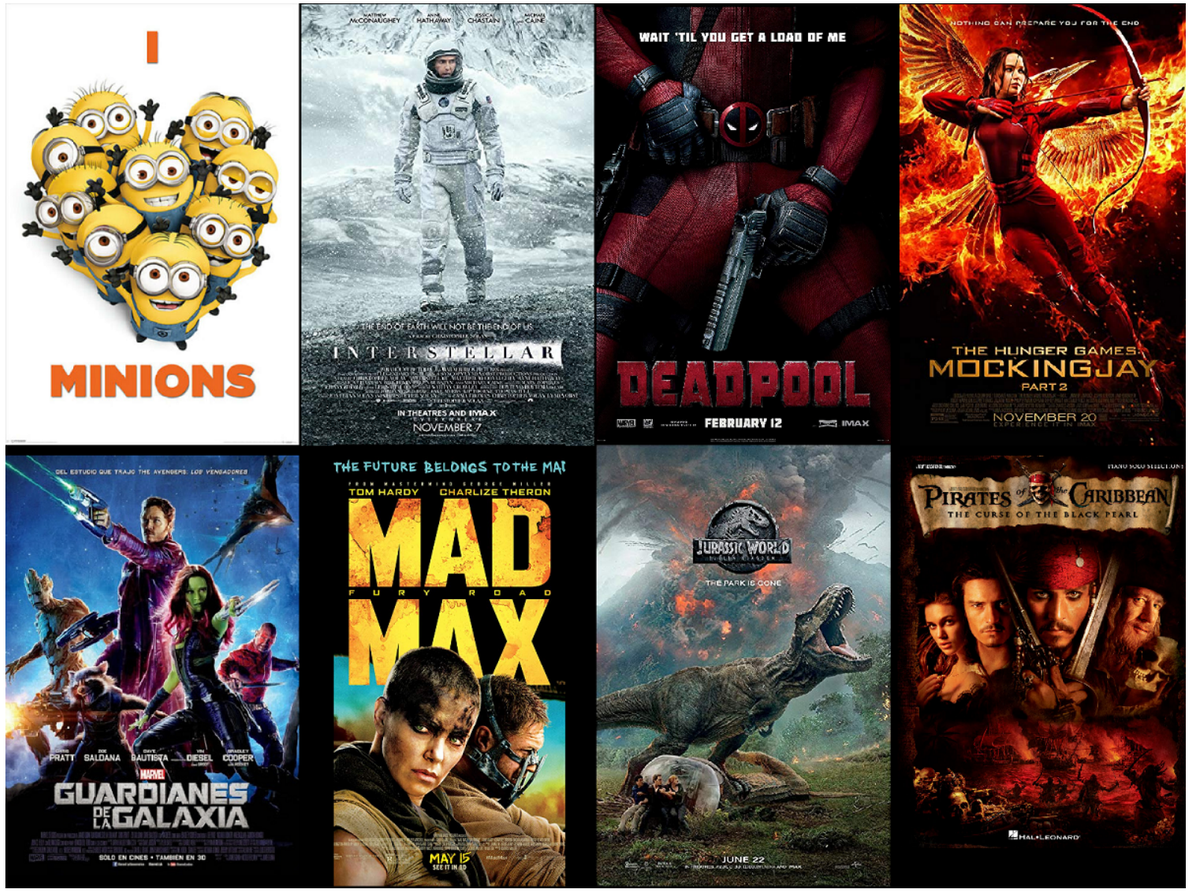
Result Table

A	B	C	D
A2	B2	C2	D2
A3	B3	C3	
A4	B4	C4	D4

4. Left join
Here we have two tables named left and right. We want to use a left join to merge them on key column C. A left join returns all of the rows from the left table and only those rows from the right table where column C matches in both. Notice the second row of the merged table. The columns from the left table are filled in, while the column from the right table is not since there wasn't a match found for that row in the right table.

New dataset

THE MOVIE DB



Movies table

```
movies = pd.read_csv('tmdb_movies.csv')
print(movies.head())
print(movies.shape)
```

```
   id     original_title   popularity      release_date
0  257    Oliver Twist     20.415572       2005-09-23
1  14290  Better Luck ...  3.877036        2002-01-12
2  38365  Grown Ups        38.864027       2010-06-24
3  9672   Infamous         3.6808959999...  2006-11-16
4  12819  Alpha and Omega  12.300789       2010-09-17
(4803, 4)
```

6. Movies table
Our first table, named movies, holds information about individual movies such as the title name and its popularity. Additionally, each movie is given an ID number. Our table starts with 4,803 rows of data.

Tagline table

```
taglines = pd.read_csv('tmdb_taglines.csv')
print(taglines.head())
print(taglines.shape)
```

```
   id       tagline
0 19995    Enter the World of Pandora.
1 285      At the end of the world, the adventure begins.
2 206647   A Plan No One Escapes
3 49026    The Legend Ends
4 49529    Lost in our world, found in another.
(3955, 2)
```

7. Tagline table
Our second table is named taglines, which contains a movie ID number and the tag line for the movie. Notice that this table has almost 4,000 rows of data, so it contains fewer movies than the movies table.

Merge with left join

```
movies_taglines = movies.merge(taglines, on='id', how='left')
print(movies_taglines.head())
```

```
   id     original_title   popularity       release_date  tagline
0  257    Oliver Twist     20.415572        2005-09-23    NaN
1  14290  Better Luck ...  3.877036         2002-01-12    Never undere...
2  38365  Grown Ups        38.864027        2010-06-24    Boys will be...
3  9672   Infamous         3.6808959999...  2006-11-16    There's more...
4  12819  Alpha and Omega  12.300789        2010-09-17    A Pawsome 3D...
```

8. Merge with left join
To merge these two tables with a left join, we use our merge method similar to what we learned in chapter 1. Here we list the movie table first and merge it to the taglines table on the ID column in both tables. However, notice an additional argument named 'how'. This argument defines how to merge the two tables. In this case, we use 'left' for a left join. The default value for how is 'inner', so we didn't need to specify this in Chapter 1 since we were only working with inner joins. The result of the merge shows a table with all of the rows from the movies table and a value for tag line where the ID column matches in both tables. Wherever there isn't a matching ID in the taglines table, a null value is entered for the tag line. Remember that pandas uses NaN to denote missing data.

Number of rows returned

```
print(movies_taglines.shape)
```

```
(4805, 5)
```

9. Number of rows returned
After the merge, our resulting table has **4,805 rows**. This is because we are returning all of the rows of data from the movies table, and the relationship between the movies table and taglines is one-to-one. Therefore, in a one-to-one merge like this one, a left join will always return the same number rows as the left table.

Note: When performing a left join, the .merge() method returns a row full of null values for columns in the right table if the key column does not have a matching value in both tables.

Note: A left join will return all of the rows from the left table. If those rows in the left table match multiple rows in the right table, then all of those rows will be returned. Therefore, the returned rows must be equal to if not greater than the left table. Knowing what to expect is useful in troubleshooting any suspicious merges.

Let's practice!

JOINING DATA WITH PANDAS

Other joins

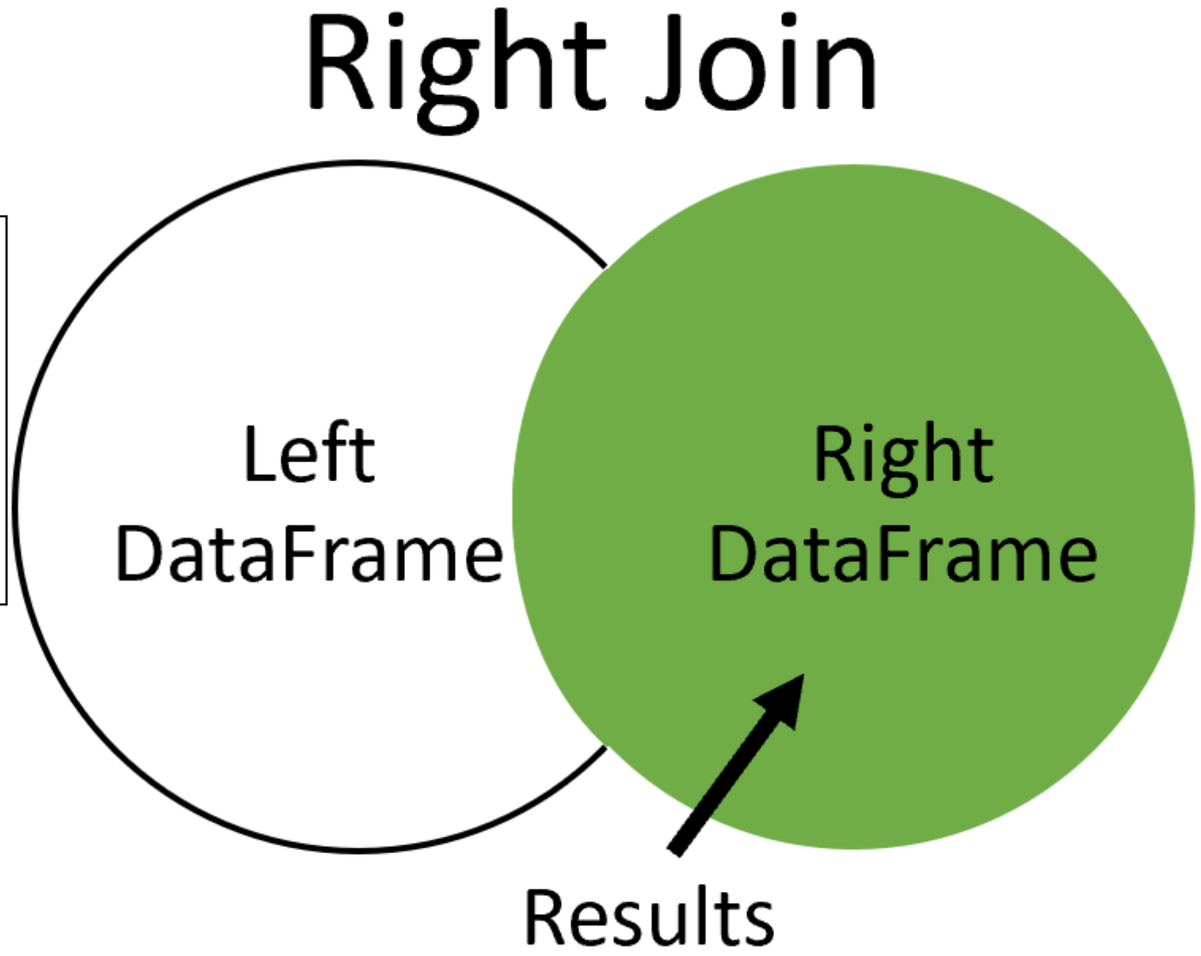
JOINING DATA WITH PANDAS

Aaren Stubberfield
Instructor

datacamp

Right join

2. Right join
Right join will return all of the rows from the right table and includes only those rows from the left table that have matching values. It is the mirror opposite of the left join.



Right join

Left Table

A	B	C
A2	B2	C2
A3	B3	C3
A4	B4	C4

Right Table

C	D
C1	D1
C2	D2
C4	D4
C5	D5

Result Table

A	B	C	D
		C1	D1
A2	B2	C2	D2
A4	B4	C4	D4
		C5	D5

Looking at data

```
movie_to_genres = pd.read_csv('tmdb_movie_to_genres.csv')
tv_genre = movie_to_genres[movie_to_genres['genre'] == 'TV Movie']
print(tv_genre)
```

```
       movie_id  genre
4998   10947     TV Movie
5994   13187     TV Movie
7443   22488     TV Movie
10061  78814     TV Movie
10790  153397    TV Movie
10835  158150    TV Movie
11096  205321    TV Movie
11282  231617    TV Movie
```

4. Looking at data
For this lesson, let's look at another table called movie_to_genres. Movies can have multiple genres, and this table lists different genres for each movie.

Filtering the data

```
m = movie_to_genres['genre'] == 'TV Movie'
tv_genre = movie_to_genres[m]
print(tv_genre)
```

```
       movie_id  genre
4998   10947     TV Movie
5994   13187     TV Movie
7443   22488     TV Movie
10061  78814     TV Movie
10790  153397    TV Movie
10835  158150    TV Movie
11096  205321    TV Movie
11282  231617    TV Movie
```

5. Filtering the data
For our right join example, let's take a sample of this data subsetting to develop a table of movies from the TV Movie genre.

Data to merge

	id	title	popularity	release_date
0	257	Oliver Twist	20.415572	2005-09-23
1	14290	Better Luck ...	3.877036	2002-01-12
2	38365	Grown Ups	38.864027	2010-06-24
3	9672	Infamous	3.6808959999...	2006-11-16
4	12819	Alpha and Omega	12.300789	2010-09-17

	movie_id	genre
4998	10947	TV Movie
5994	13187	TV Movie
7443	22488	TV Movie
10061	78814	TV Movie
10790	153397	TV Movie

6. Data to merge
Our goal is to merge it with the movies table. We will set movies as our left table and merge it with the tv_genre table. We want to use a right join to check that our movies table is not missing data. In addition to showing a right join, this example also allows us to look at another feature. Notice that the column with the movie ID number in the movies table is named id, and in the tv_genre table it is named movie_id. **The merge method has a feature to take this into account.**

Merge with right join

```
tv_movies = movies.merge(tv_genre, how='right',
                         left_on='id', right_on='movie_id')
print(tv_movies.head())
```

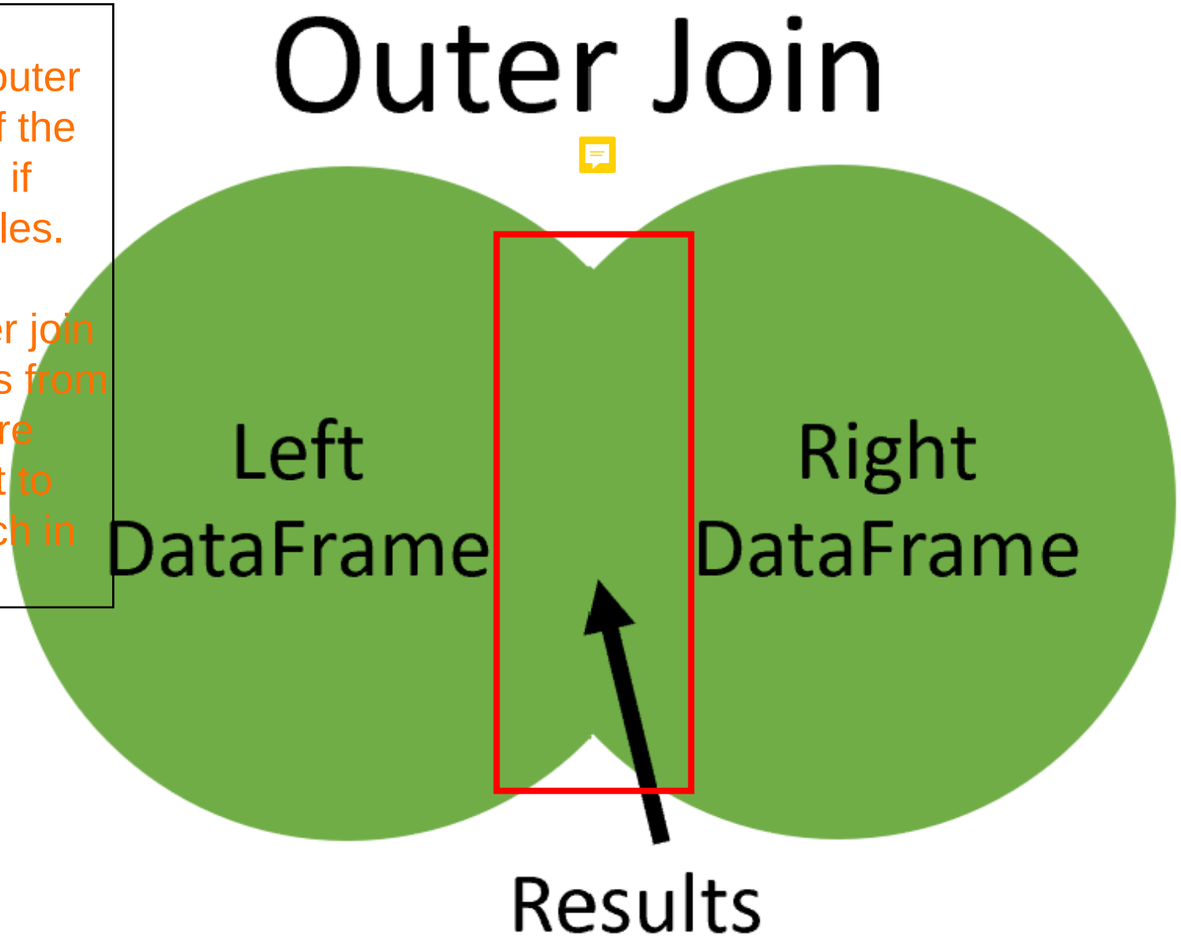
	id	title	popularity	release_date	movie_id	genre
0	153397	Restless	0.812776	2012-12-07	153397	TV Movie
1	10947	High School ...	16.536374	2006-01-20	10947	TV Movie
2	231617	Signed, Seal...	1.444476	2013-10-13	231617	TV Movie
3	78814	We Have Your...	0.102003	2011-11-12	78814	TV Movie
4	158150	How to Fall ...	1.923514	2012-07-21	158150	TV Movie

7. Merge with right join
The code for this merge has some new elements. First of all, we set the how argument to right so that the merge performs a right join. Additionally, we introduce two new arguments, named left_on and right_on. They allow us to tell the merge which key columns from each table to merge the tables. We list movies as the left table, so we set left_on to id and right_on to movie_id. Our returned table has movies that match our table of tv_genres. There does not appear to be any null values in the columns from the movies table.

Outer join

8. Outer join
Our last type of join is called an outer join. An outer join will return all of the rows from both tables regardless if there is a match between the tables.

One cool aspect of using an outer join is that, because it returns all rows from both merged tables and null where they do not match, you can use it to find rows that do not have a match in the other table.



Outer join

Left Table

A	B	C
A2	B2	C2
A3	B3	C3
A4	B4	C4

Right Table

C	D
C1	D1
C2	D2
C4	D4
C5	D5

Result Table

A	B	C	D
		C1	D1
A2	B2	C2	D2
A3	B3	C3	
A4	B4	C4	D4
		C5	D5

9. Outer join
Here is a simple example of an outer join. Where the key column used to join the tables has no match, null values are returned. That is why in the result, the columns from the left table are missing in rows one and five, and in column D row three is missing.

Datasets for outer join

```
m = movie_to_genres['genre'] == 'Family'
family = movie_to_genres[m].head(3)
```

```
m = movie_to_genres['genre'] == 'Comedy'
comedy = movie_to_genres[m].head(3)
```

	movie_id	genre
0	12	Family
1	35	Family
2	105	Family

	movie_id	genre
0	5	Comedy
1	13	Comedy
2	35	Comedy

10. Datasets for outer join
For an example of this, we filter the movie_to_genres table as before into two very small tables. One table has data on Family movies, and the other has Comedy movies.

Merge with outer join

```
family_comedy = family.merge(comedy, on='movie_id', how='outer',
                             suffixes=('_fam', '_com'))

print(family_comedy)
```

```
   movie_id  genre_fam  genre_com
0 12         Family     NaN
1 35         Family     Comedy
2 105        Family     NaN
3 5          NaN        Comedy
4 13         NaN        Comedy
```

11. Merge with outer join
In this merge, we list the family table as the left table and merge it on the movie_id column. The how argument is set to outer for an outer join. Both of our tables have the same column names. Therefore, we add suffixes to show what table the columns originated. In our result table, every row is returned for both tables and we see some null values. In our original comedy tables ID number 12 does not exist. Therefore a null is shown. Similarly, in our last row, movie ID 13 wasn't in the family dataset so it has a null.

Note: Using an outer join, you were able to pick only those rows where the actor played in only one of the two movies.

Let's practice!

JOINING DATA WITH PANDAS

Merging a table to itself

JOINING DATA WITH PANDAS

Aaren Stubberfield
Instructor

datacamp

Sequel movie data

```
print(sequel.head())
```

```
   id      title          sequel
0 19995   Avatar         NaN
1 862     Toy Story      863
2 863     Toy Story 2    10193
3 597     Titanic        NaN
4 24428   The Avengers   NaN
```

2. Sequel movie data
So when would you ever need to merge a table to itself? The table shown here is called sequels and has three columns. It contains a column for movie id, title, and sequel. The sequel number refers to the movie id that is a sequel to the original movie. For example, in the second row the movie is titled Toy Story, and has an id equal to 862. The sequel number of this row is 863. This is the movie id for Toy Story 2, the sequel to Toy Story. If we continue, 10193 is the movie id Toy Story 3 which is the sequel for Toy Story 2.

Merging a table to itself

Left Table

id	title	sequel
19995	Avatar	
862	Toy Story	863
863	Toy Story 2	10193
597	Titanic	
24428	The Ave...	

Right Table

id	title	sequel
19995	Avatar	
862	Toy Story	863
863	Toy Story 2	10193
597	Titanic	
24428	The Ave...	

Merge Columns

Result Table

id_x	title_x	sequel_x	id_y	title_y	sequel_y
862	Toy Story	863	863	Toy Story 2	10193
863	Toy Story 2	10193	10193	Toy Story 3	

3. Merging a table to itself

If we would like to see a table with the movies and the corresponding sequel movie in one row of the table, we will need to merge the table to itself. In the left table, the sequel ID for Toy Story of 863 is matched with 863 in the ID column of the right table. Similarly, Toy Story 2 of the left table is matched with Toy Story 3 in the right table. We will talk more about this later, but the merge is an inner join. Therefore, we do not see Avatar and Titanic because they do not have sequels.

Merging a table to itself

```
original_sequels = sequels.merge(sequels, left_on='sequel', right_on='id',
                                 suffixes=('_org','_seq'))
print(original_sequels.head())
```

```
   id_org  title_org          sequel_org  id_seq  title_seq          sequel_seq
0  862     Toy Story          863         863     Toy Story 2        10193
1  863     Toy Story 2        10193       10193   Toy Story 3        NaN
2  675     Harry Potter...    767         767     Harry Potter...    NaN
3  121     The Lord of ...    122         122     The Lord of ...    NaN
4  120     The Lord of ...    121         121     The Lord of ...    122
```

4. Merging a table to itself

We can think of it as merging two copies of the same table. All of the aspects we have reviewed regarding merging two tables still apply here. Therefore, we can merge the tables on different columns. We'll use the 'left_on' and 'right_on' attributes to match rows where the sequel's id matches the original movie's id. Finally, setting the suffixes argument in the merge method allows us to identify which columns describe the original movie and which describe the sequel. When we look at the results of the merge, the 'title_org' and 'title_seq' list the original and sequel movies, respectively. Here we listed the original movie and its sequel in one row.

Continue format results

```
print(original_sequels[,['title_org','title_seq']].head())
```

```
  title_org          title_seq
0 Toy Story          Toy Story 2
1 Toy Story 2        Toy Story 3
2 Harry Potter...    Harry Potter...
3 The Lord of ...    The Lord of ...
4 The Lord of ...    The Lord of ...
```

5. Continue format results
Now that we have our result table from the merge, we could select only the `title_org`, and `title_seq` columns, and we can see that we've successfully linked each movie to its sequel.

Merging a table to itself with left join

```
original_sequels = sequels.merge(sequels, left_on='sequel', right_on='id',
                                 how='left', suffixes=('_org','_seq'))
print(original_sequels.head())
```

```
   id_org  title_org      sequel_org  id_seq  title_seq     sequel_seq
0  19995   Avatar         NaN         NaN     NaN           NaN
1  862     Toy Story      863         863     Toy Story 2   10193
2  863     Toy Story 2    10193       10193   Toy Story 3   NaN
3  597     Titanic        NaN         NaN     NaN           NaN
4  24428   The Avengers   NaN         NaN     NaN           NaN
```

6. Merging a table to itself with left join
... we can use the different types of joins we have already reviewed. Let's take the same merge from earlier but make it a left join. The 'how' argument is set in the merge method to left from the default 'inner'. Now the resulting table will show all of our original movie info. If the sequel movie exists in the table, it will fill out the rest of the row. If you compare this to our earlier merger, you now see movies like Avatar and Titanic in the result set.

When to merge at table to itself

Common situations:

- Hierarchical relationships
- Sequential relationships
- Graph data

7. When to merge at table to itself
You might need to merge a table to itself when working with tables that have a hierarchical relationship, like employee and manager.

You might use this on sequential relationships such as logistic movements.

Graph data, such as networks of friends, might also require this technique.

Let's practice!

JOINING DATA WITH PANDAS

Merging on indexes

JOINING DATA WITH PANDAS

Aaren Stubberfield
Instructor

1. Merging on indexes
So far, we've only looked at merging two tables together using their columns. In this lesson, we'll discuss how to merge tables using their indexes. Often, the DataFrame indexes are given a unique id that we can use when merging two tables together.

Table with an index

```
  id     title            popularity  release_date
0 257    Oliver Twist     20.415572   2005-09-23
1 14290  Better Luck ...  3.877036    2002-01-12
2 38365  Grown Ups        38.864027   2010-06-24
3 9672   Infamous         3.680896    2006-11-16
4 12819  Alpha and Omega  12.300789   2010-09-17
```

```
       title            popularity  release_date
id
257    Oliver Twist     20.415572   2005-09-23
14290  Better Luck ...  3.877036    2002-01-12
38365  Grown Ups        38.864027   2010-06-24
9672   Infamous         3.680896    2006-11-16
12819  Alpha and Omega  12.300789   2010-09-17
```

Setting an index

```
movies = pd.read_csv('tmdb_movies.csv', index_col=['id'])
print(movies.head())
```

```
        title              popularity  release_date
id
257     Oliver Twist       20.415572   2005-09-23
14290   Better Luck ...    3.877036    2002-01-12
38365   Grown Ups          38.864027   2010-06-24
9672    Infamous           3.680896    2006-11-16
12819   Alpha and Omega    12.300789   2010-09-17
```

3. Setting an index
There are different methods to set the index of a table, but if our data starts off in a CSV file, we can use the index_col argument of the read_csv method. This lesson will not focus on how to set a table index, but how to use that index to merge two tables together.

Merge index datasets

```
       title             popularity  release_date
id
257    Oliver Twist      20.415572   2005-09-23
14290  Better Luck ...   3.877036    2002-01-12
38365  Grown Ups         38.864027   2010-06-24
9672   Infamous          3.680896    2006-11-16
```

```
        tagline
id
19995   Enter the Wo...
285     At the end o...
206647  A Plan No On...
49026   The Legend Ends
```

4. Merge index datasets
Recall our example to merge the movies and taglines tables using the id column with a left join. Let's recreate that merge using the index which is now the id for tables.

Merging on index

Note: Merging on indexes is just like merging on columns, so if you need to merge based on indexes, there's no need to turn the indexes into columns first.

```
movies_taglines = movies.merge(taglines, on='id', how='left')
print(movies_taglines.head())
```

```
       title            popularity  release_date  tagline
id
257    Oliver Twist     20.415572   2005-09-23    NaN
14290  Better Luck ...  3.877036    2002-01-12    Never undere...
38365  Grown Ups        38.864027   2010-06-24    Boys will be...
9672   Infamous         3.680896    2006-11-16    There's more...
12819  Alpha and Omega  12.300789   2010-09-17    A Pawsome 3D...
```

5. Merging on index
Our merge statement looks identical to before. However, in this case we are inputting to the 'on' argument the index level name which is called 'id'. The merge method automatically adjusts to accept index names or column names. The returned table looks as before, except the 'id' is the index.

MultiIndex datasets

```
samuel = pd.read_csv('samuel.csv',
                     index_col=['movie_id',
                                'cast_id'])
print(samuel.head())
```

```
                    name
movie_id  cast_id
184       3         Samuel L. Jackson
319       13        Samuel L. Jackson
326       2         Samuel L. Jackson
329       138       Samuel L. Jackson
393       21        Samuel L. Jackson
```

```
casts = pd.read_csv('casts.csv',
                    index_col=['movie_id',
                               'cast_id'])
print(casts.head())
```

```
                    character
movie_id  cast_id
5         22        Jezebel
          23        Diana
          24        Athena
          25        Elspeth
          26        Eva
```

6. MultiIndex datasets
Let's try a multiIndex merge. Here, we have two tables with a multiIndex that holds the movie ID and cast ID. The first table, named 'samuel', has the movie and cast ID for a group of movies that Samuel L. Jackson acted in. The second table, named cast, has the movie ID and cast ID for a number of movie characters. Let's merge these two tables on their multiIndex.

MultiIndex merge

```
samuel_casts = samuel.merge(casts, on=['movie_id','cast_id'])
print(samuel_casts.head())
print(samuel_casts.shape)
```

```
                    name                 character
movie_id  cast_id
184       3         Samuel L. Jackson  Ordell Robbie
319       13        Samuel L. Jackson  Big Don
326       2         Samuel L. Jackson  Neville Flynn
329       138       Samuel L. Jackson  Arnold
393       21        Samuel L. Jackson  Rufus
(67, 2)
```

7. MultiIndex merge
In this merge, we pass in a list of index level names to the 'on' argument, just like we did when merging on multiple columns. Since this is an inner join, both the movie_id and cast_id must match in each table to be returned in the result. It's interesting to see that Samuel Jackson has acted in over 65 movies! That's a lot.

Index merge with left_on and right_on

```
       title             popularity  release_date
id
257    Oliver Twist      20.415572   2005-09-23
14290  Better Luck ...   3.877036    2002-01-12
38365  Grown Ups         38.864027   2010-06-24
9672   Infamous          3.680896    2006-11-16
```

8. Index merge with left_on and right_on
There is one more thing regarding merging on indexes. If the index level names are different between the two tables that we want to merge, then we can use the left_on and right_on arguments of the merge method. Let's go back to our movies table, shown in the top panel, and merge it with our movies_to_genres table, shown in the lower panel.

```
           genre
movie_id
5          Crime
5          Comedy
11         Science Fiction
11         Action
```

Index merge with left_on and right_on

```
movies_genres = movies.merge(movie_to_genres, left_on='id', left_index=True,
                             right_on='movie_id', right_index=True)
print(movies_genres.head())
```

```
    id  title        popularity  release_date  genre
5    5  Four Rooms   22.876230   1995-12-09    Crime
5    5  Four Rooms   22.876230   1995-12-09    Comedy
11  11  Star Wars    126.393695  1977-05-25    Science Fiction
11  11  Star Wars    126.393695  1977-05-25    Action
11  11  Star Wars    126.393695  1977-05-25    Adventure
```

9. Index merge with left_on and right_on
In this merge, since we list the movies table as the left table, we set left_on equal to id and right_on equal to movie_id. Additionally, since we are merging on indexes, we need to set left_index and right_index to True. These arguments take only True or False. Whenever we are using the left_on or right_on arguments with an index, we need to set the respective left_index and right_index arguments to True. The left_index and right_index tell the merge method to use the separate indexes.

Let's practice!

JOINING DATA WITH PANDAS