Join Using Pandas - 1

Inner Join

Merging tables

	ward	alderman	address	zip
0	1	Proco "Joe" ...	2058 NORTH W...	60647
1	2	Brian Hopkins	1400 NORTH ...	60622
2	3	Pat Dowell	5046 SOUTH S...	60609
3	4	William D. B...	435 EAST 35T...	60616
4	5	Leslie A. Ha...	2325 EAST 71...	60649

	ward	pop_2000	pop_2010	change	address	zip
0	1	52951	56149	6%	2765 WEST SA...	60647
1	2	54361	55805	3%	WM WASTE MAN...	60622
2	3	40385	53039	31%	17 EAST 38TH...	60653
3	4	51953	54589	5%	31ST ST HARB...	60653
4	5	55302	51455	-7%	JACKSON PARK...	60637

- To merge the above tables, we will perform the merge/join operation on the ward column, as ward is the primary key for the above two tables.
- After merging ward 1 will have Proco Joe, … 52951
- As the above two tables/dataframes have address and zip columns after merging, these will be distinguished with the subscript address_x, address_y.
- wards.merge(census, on = ‘ward’)
- Merger wards and census dataframes on the ward column.

Inner join

```
wards_census = wards.merge(census, on='ward')
print(wards_census.head(4))
```

	ward	alderman	address_x	zip_x	pop_2000	pop_2010	change	address_y	zip_y
0	1	Proco "Joe" ...	2058 NORTH W...	60647	52951	56149	6%	2765 WEST SA...	60647
1	2	Brian Hopkins	1400 NORTH ...	60622	54361	55805	3%	WM WASTE MAN...	60622
2	3	Pat Dowell	5046 SOUTH S...	60609	40385	53039	31%	17 EAST 38TH...	60653
3	4	William D. B...	435 EAST 35T...	60616	51953	54589	5%	31ST ST HARB...	60653

```
print(wards_census.shape)
```

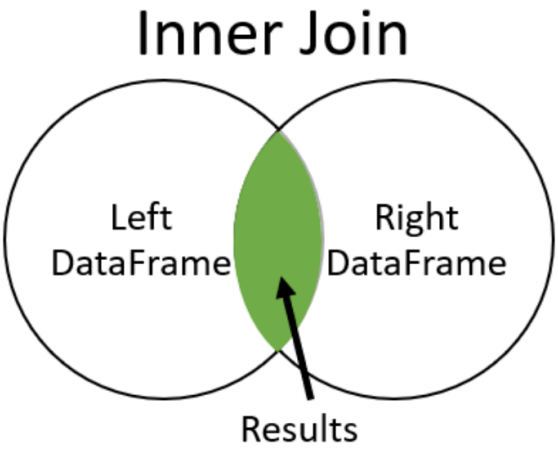
```
(50, 9)
```

The above merging/joining operation performed is called Inner join. Inner Join will return only those rows, that have the matching column values in both the tables.

toystory_tag = toy_story.merge(taglines, on = 'id', how = 'inner')

Here only those rows were returned, for which data of the ward column matched in the ward and census tables

Inner join



Suffixes

```
print(wards_census.columns)
```

```
Index(['ward', 'alderman', 'address_x', 'zip_x', 'pop_2000', 'pop_2010', 'change',
       'address_y', 'zip_y'],
      dtype='object')
```

By default x and y are used as the suffix when both the tables merged have the same column names.

Suffixes

```
wards_census = wards.merge(census, on='ward', suffixes=('_ward','_cen'))
print(wards_census.head())
print(wards_census.shape)
```

```
  ward  alderman         address_ward     zip_ward  pop_2000  pop_2010  change  address_cen      zi
0 1     Proco "Joe" ...  2058 NORTH W...  60647     52951     56149     6%      2765 WEST SA...  60
1 2     Brian Hopkins    1400 NORTH  ...  60622     54361     55805     3%      WM WASTE MAN...  60
2 3     Pat Dowell       5046 SOUTH S...  60609     40385     53039     31%     17 EAST 38TH...  60
3 4     William D. B...  435 EAST 35T...  60616     51953     54589     5%      31ST ST HARB...  60
4 5     Leslie A. Ha...  2325 EAST 71...  60649     55302     51455     -7%     JACKSON PARK...  60
(50, 9)
```

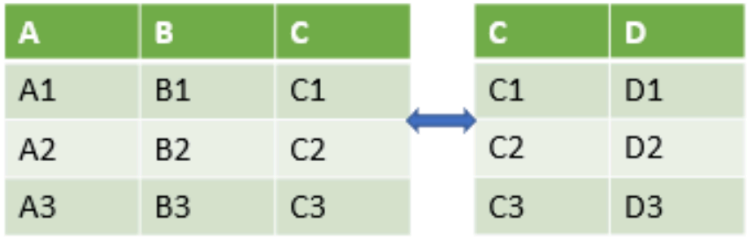
- We can pass in the suffixes inside the merge method to define suffixes for the columns

Example:
# Merge the taxi_owners and taxi_veh tables setting a suffix
taxi_own_veh = taxi_owners.merge(taxi_veh, on='vid', suffixes=('_own','_veh'))

One-to-many relationships

one to one relationship

One-to-one



A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3

C	D
C1	D1
C2	D2
C3	D3

One-To-One = Every row in the left table is related to only one row in the right table

every row in the ward table is related to one row in the census table

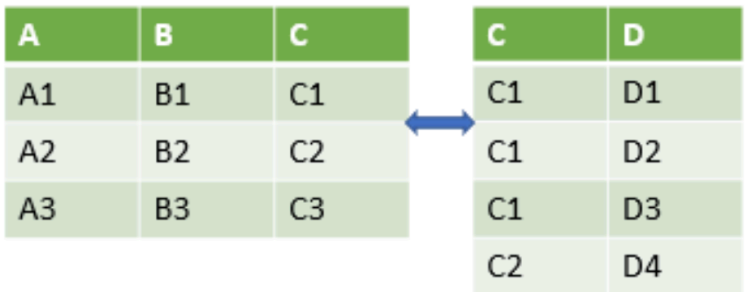
One-to-one example

	ward	alderman	address	zip
0	1	Proco "Joe" ...	2058 NORTH W...	60647
1	2	Brian Hopkins	1400 NORTH ...	60622
2	3	Pat Dowell	5046 SOUTH S...	60609
3	4	William D. B...	435 EAST 35T...	60616
4	5	Leslie A. Ha...	2325 EAST 71...	60649

	ward	pop_2000	pop_2010	change	address	zip
0	1	52951	56149	6%	2765 WEST SA...	60647
1	2	54361	55805	3%	WM WASTE MAN...	60622
2	3	40385	53039	31%	17 EAST 38TH...	60653
3	4	51953	54589	5%	31ST ST HARB...	60653
4	5	55302	51455	-7%	JACKSON PARK...	60637

One-to-many relationship

One-to-many

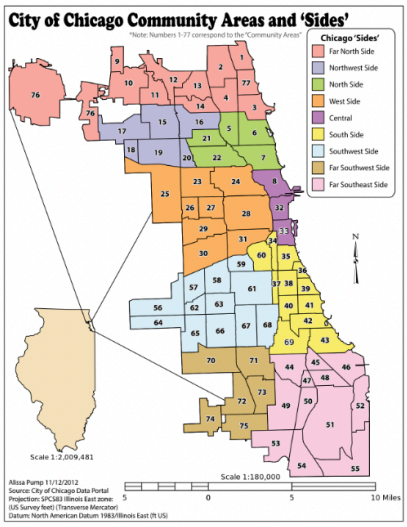


A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3

C	D
C1	D1
C1	D2
C1	D3
C2	D4

One-To-Many = Every row in left table is related to one or more rows in the right table

One-to-many example



In every ward there are multiple businesses, we will merge the wards table with the business table, wherein business table will have multiple rows for one ward.

One-to-many example

	ward	alderman	address	zip
0	1	Proco "Joe" ...	2058 NORTH W...	60647
1	2	Brian Hopkins	1400 NORTH ...	60622
2	3	Pat Dowell	5046 SOUTH S...	60609
3	4	William D. B...	435 EAST 35T...	60616
4	5	Leslie A. Ha...	2325 EAST 71...	60649

	account	ward	aid	business	address	zip
0	307071	3	743	REGGIE'S BAR...	2105 S STATE ST	60616
1	10	10	829	HONEYBEERS	13200 S HOUS...	60633
2	10002	14	775	CELINA DELI	5089 S ARCHE...	60632
3	10005	12	nan	KRAFT FOODS ...	2005 W 43RD ST	60609
4	10044	44	638	NEYBOUR'S TA...	3651 N SOUTH...	60613

One-to-many example

```
ward_licenses = wards.merge(licenses, on='ward', suffixes=('_ward','_lic'))
print(ward_licenses.head())
```

```
  ward  alderman         address_ward     zip_ward  account  aid  business         address_lic
0 1     Proco "Joe" ...  2058 NORTH W...  60647     12024    nan  DIGILOG ELEC...  1038 N ASHLA...
1 1     Proco "Joe" ...  2058 NORTH W...  60647     14446    743  EMPTY BOTTLE...  1035 N WESTE...
2 1     Proco "Joe" ...  2058 NORTH W...  60647     14624    775  LITTLE MEL'S...  2205 N CALIF...
3 1     Proco "Joe" ...  2058 NORTH W...  60647     14987    nan  MR. BROWN'S ...  2301 W CHICA...
4 1     Proco "Joe" ...  2058 NORTH W...  60647     15642    814  Beat Kitchen     2000-2100 W ...
```

For the case of one to many relationships. pandas will take care of the merging as shown in the above example.

One-to-many example

```
print(wards.shape)
```

```
(50, 4)
```

```
print(ward_licenses.shape)
```

```
(10000, 9)
```

When we have one to many relationships in the tables, the number of rows returned in the merged table will be different.

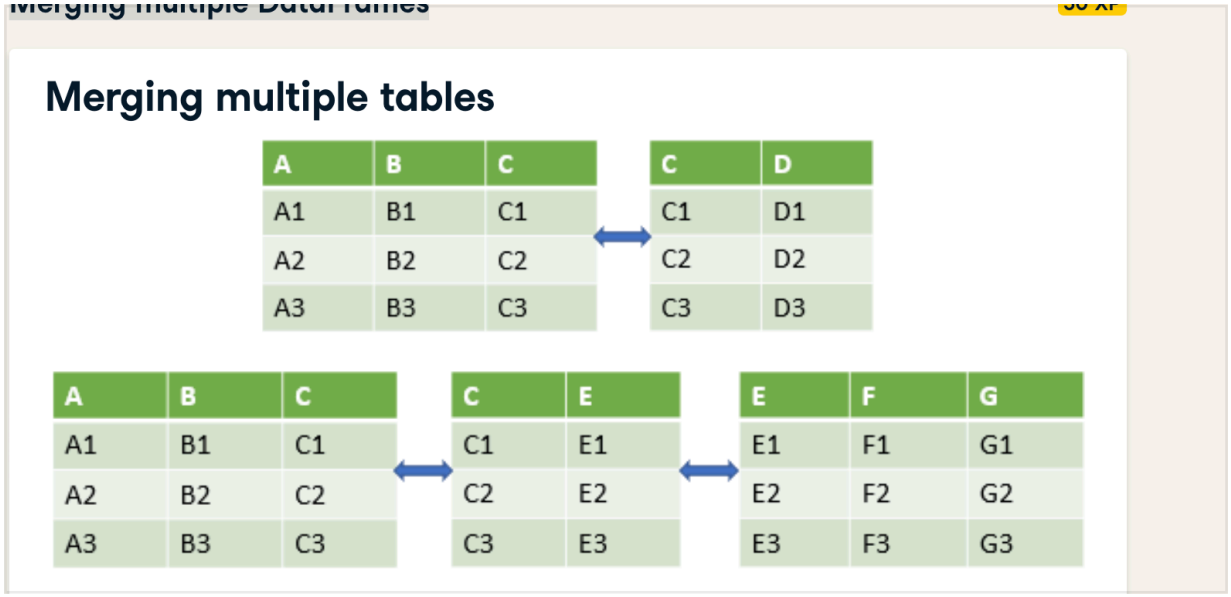
# Merge the licenses and biz_owners table on account
licenses_owners = licenses.merge(biz_owners, on = 'account')

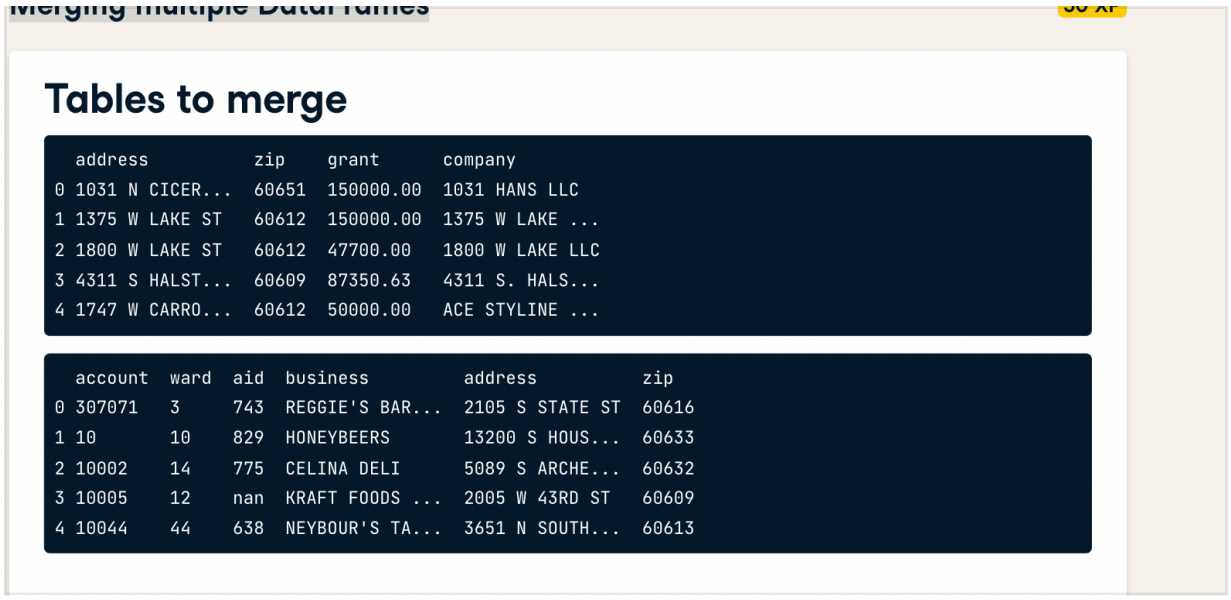
# Group the results by title then count the number of accounts
counted_df = licenses_owners.groupby('title').agg({'account':'count'})

# Sort the counted_df in desending order
sorted_df = counted_df.sort_values(by = 'account', ascending = False)

```
# Use .head() method to print the first few rows of sorted_df
print(sorted_df.head())
```

Merging multiple DataFrames





Single merge

```
grants.merge(licenses, on=['address','zip'])
```

	address	zip	grant	company	account	ward	aid	business
0	1020 N KOLMA...	60651	68309.8	TRITON INDUS...	7689	37	929	TRITON INDUS...
1	10241 S COMM...	60617	33275.5	SOUTH CHICAG...	246598	10	nan	SOUTH CHICAG...
2	11612 S WEST...	60643	30487.5	BEVERLY RECO...	3705	19	nan	BEVERLY RECO...
3	1600 S KOSTN...	60623	128513.7	CHARTER STEE...	293825	24	nan	LEELO STEEL,...
4	1647 W FULTO...	60612	5634.0	SN PECK BUIL...	85595	27	673	S.N. PECK BU...

As there was a chance that address and zip could be repeated, we merge on the combination of both.
We are specifying that in order for the columns to be merged, we want the address and zip from the left column match to the address of the right column.

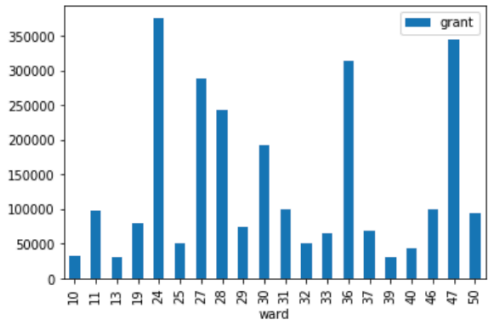
Merging multiple tables

```
grants_licenses_ward = grants.merge(licenses, on=['address','zip']) \
                        .merge(wards, on='ward', suffixes=('_bus','_ward'))
grants_licenses_ward.head()
```

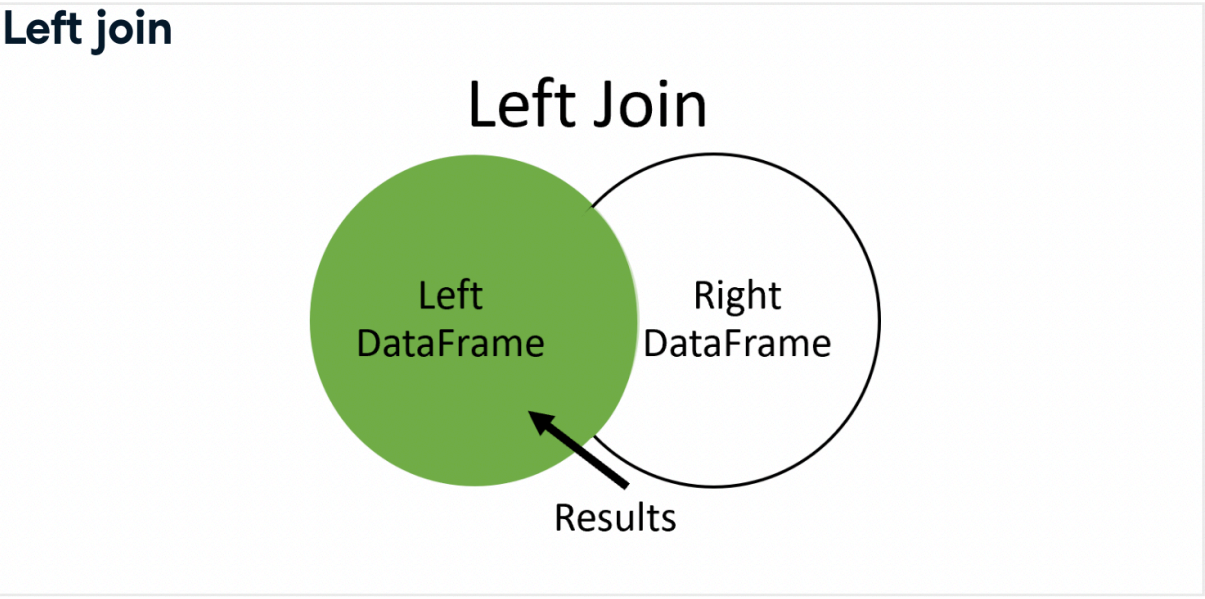
	address_bus	zip_bus	grant	company	account	ward	aid	business	alderma
0	1020 N KOLMA...	60651	68309.8	TRITON INDUS...	7689	37	929	TRITON INDUS...	Emma M.
1	10241 S COMM...	60617	33275.5	SOUTH CHICAG...	246598	10	nan	SOUTH CHICAG...	Susan S
2	11612 S WEST...	60643	30487.5	BEVERLY RECO...	3705	19	nan	BEVERLY RECO...	Matthew
3	3502 W 111TH ST	60655	50000.0	FACE TO FACE...	263274	19	704	FACE TO FACE	Matthew
4	1600 S KOSTN...	60623	128513.7	CHARTER STEE...	293825	24	nan	LEELO STEEL,...	Michael

We can merge multiple tables by specifying merge method sequentially on the tables.

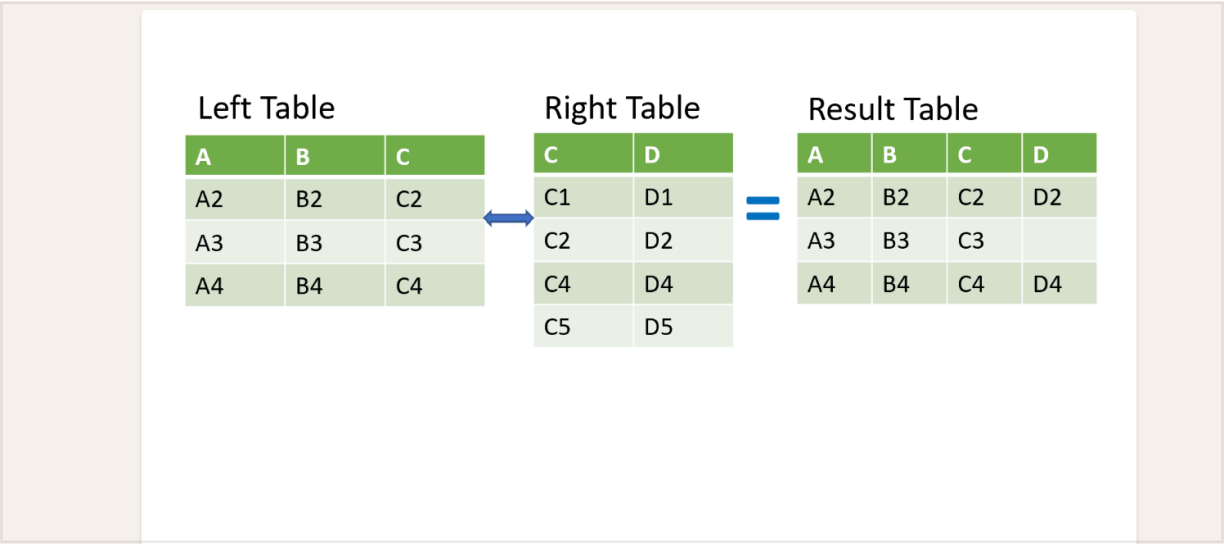
```
import matplotlib.pyplot as plt
grant_licenses_ward.groupby('ward').agg('sum').plot(kind='bar', y='grant')
plt.show()
```



Left Join



The left join operation returns the all the rows from the left table, while including only those rows on the right table that match where key columns match.



Here only C2 and C4 values are present in the right table, therefore we only merge two rows from the right table with the left table and rest of the entries for the D column where no match was found are left empty.

- We will use the pandas merge method to perform the merging of the tables, the default value for the how parameter is inner, to perform left join, we will pass in left as the value for how.

```
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...
```

In the above result we can see that NaN will be present in the tagline column for all those movies for which there was no matching id in the right table.

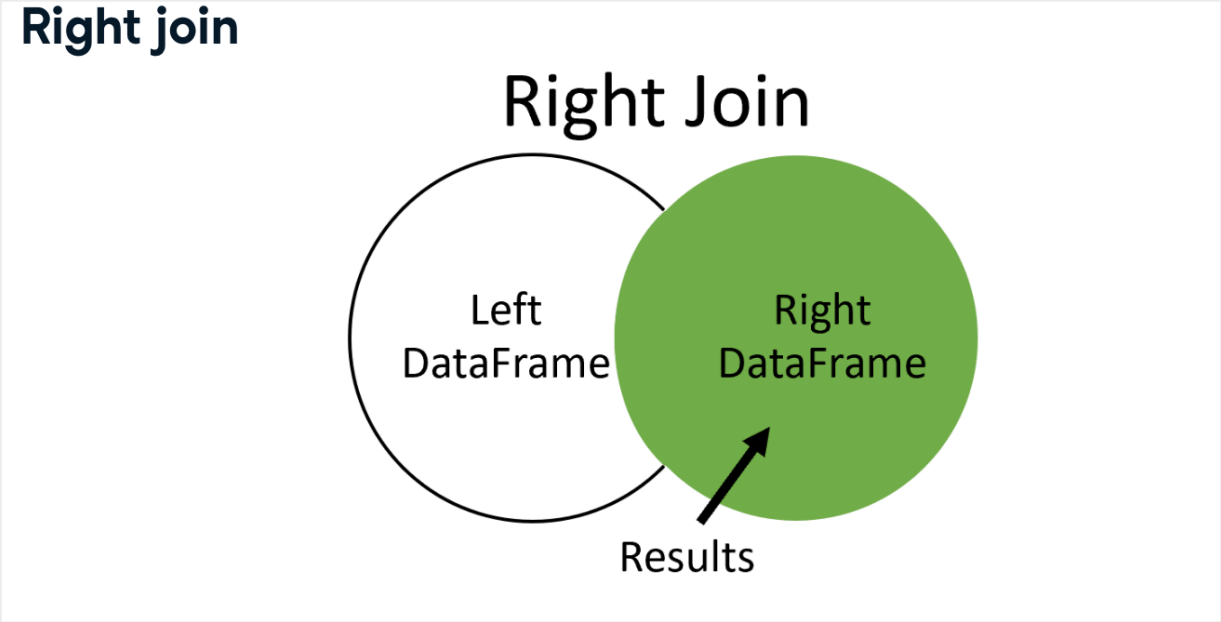
```
print(movies_taglines.shape)
```

```
(4805, 5)
```

Here the number of rows in the resultant merged/joined table will be same as those in the left table as we decide to keep all the rows in the left table, and add an additional column only where the join criteria matches.
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.

———————

Right Joins



The result of this merge operation will have all the rows from the right table, and only those rows from the left table with the matching values.
If there are multiple rows with the matching values, we will have multiple rows in the resultant table.

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

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
```

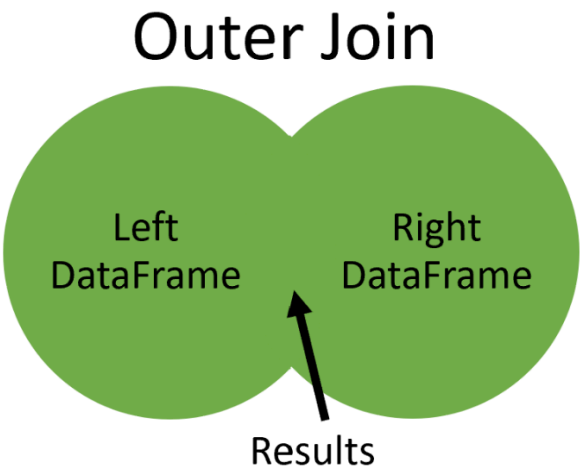
Merge with right join

```
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
```

- When merging the two columns, we notice that the columns are named differently, therefore we will make use of left_on and right_on parameters, which specify the names of the columns on the left and right tables, respectively.
- Right join is a mirrored version of left join.

Outer join



Outer join will return the rows from both the tables, regardless of whether there was a match or not.

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

Here C2 and C4 have data in all of the four columns as the C data matched, in the other rows, we can see that where there wasn't a match, the previous data is present and the new columns have null value.

Datasets for outer join

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

```
  movie_id  genre
0 12        Family
1 35        Family
2 105       Family
```

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

```
  movie_id  genre
0 5         Comedy
1 13        Comedy
2 35        Comedy
```

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
```

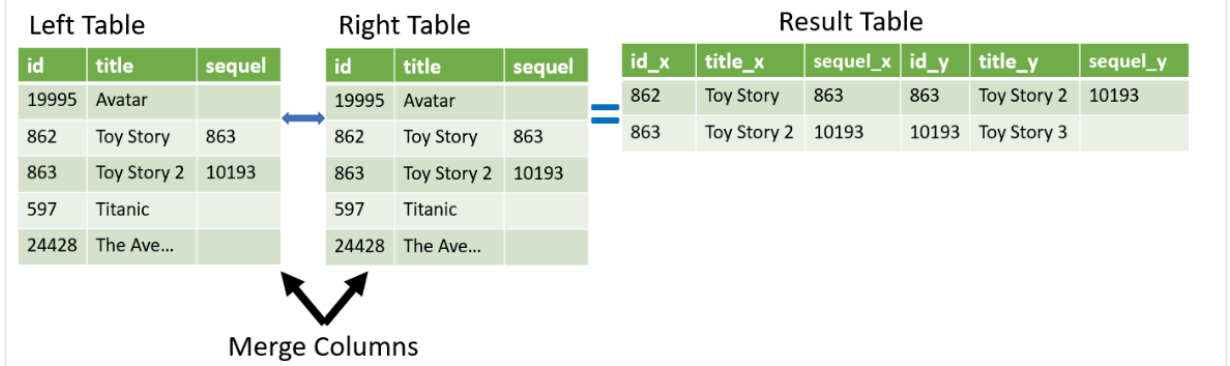
In the above result, we can see that only id 35 was present in both the tables, so only it has values in both columns, whereas other have values in either genere_fam or genre_Com.
If an inner join operation was performed we would have only 1 row, left and right joins would give us 3 rows with only id 35 row having all the values.

Merging a table to itself - Self join - Special Case of join wherein we can use either inner, left, outer, right joins.
We would want to merge the table with itself in the below scenario.
Here sequel column contains the id of the movie it is sequel to,

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
```



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

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

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	

When we want to merge table with itself to place the sequel of a movie in the same row, with the sequel column we will use the self join operation.
The merger is an inner join, therefore we do not get any rows where there isn't a match in the result.

```
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
```

In the above example we can see that we have passed in the table as the parameter to the merge method, and the left_on and right_on parameters have different values, if there
Self-Join using Left join.

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
```

We may want to merge a table to itself in the below scenarios

When to merge at table to itself

Common situations:

- Hierarchical relationships
- Sequential relationships
- Graph data

Hierarchical relationships - Employee to manager
Graph - Child Node to its parent node
Sequential relationships - Logistic Movements (From and To places)

———————————

Merging on table indexes
Here id is the index

Merging on index

```
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...
```

We can also join tables when there are nested indexes using the merge operation

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)
```

We can make use of left_on and right_on to join when we have different index columns in the tables.

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
```