Interactive Course

**Joining Data with pandas**

* 4 hours
* 15 Videos
* 52 Exercises
* 96,009 Participants
* 4,150 XP

**Course Description**

Being able to combine and work with multiple datasets is an essential skill for any aspiring Data Scientist. pandas is a crucial cornerstone of the Python data science ecosystem, with Stack Overflow recording 5 million views for pandas questions. Learn to handle multiple DataFrames by combining, organizing, joining, and reshaping them using pandas. You'll work with datasets from the World Bank and the City Of Chicago. You will finish the course with a solid skillset for data-joining in pandas.

1. 1

**Data Merging Basics**

0%

Learn how you can merge disparate data using inner joins. By combining information from multiple sources you’ll uncover compelling insights that may have previously been hidden. You’ll also learn how the relationship between those sources, such as one-to-one or one-to-many, can affect your result.

**Inner join**

50 xp

**What column to merge on?**

50 xp

**Your first inner join**

100 xp

**Inner joins and number of rows returned**

100 xp

**One-to-many relationships**

50 xp

**One-to-many classification**

100 xp

**One-to-many merge**

100 xp

**Merging multiple DataFrames**

50 xp

**Total riders in a month**

100 xp

**Three table merge**

100 xp

**Daily XP550**

# Inner join

**50 XP**

## 1. Inner join

Welcome! I am Aaren Stubberfield and I will be your instructor for this course. The pandas package is a powerful tool for manipulating and transforming data in Python. However, when working on an analysis, the data needed could be in multiple tables. This course will focus on the vital skill of merging tables together.

## 2. For clarity

As we start, two quick clarifications. First, through other courses on DataCamp, you may have learned how to import tabular data as DataFrames. In this course, you may hear the words table and DataFrame, but they are equivalent here. Second, we will refer to combining different tables together as merging tables, but note that some refer to this same process as joining.

1. 1 Photo by David Travis on Unsplash

## 3. Chicago data portal dataset

To help us learn about merging tables, we will use data from the city of Chicago data portal.

1. 1 Photo by Pedro Lastra on Unsplash

## 4. Datasets for example

The city of Chicago is divided into fifty local neighborhoods called wards. We have a table with data about the local government offices in each ward. In this example, we want to merge the local government data with census data about the population of each ward.

1. 1 Ward image By Alissapump, Own work, CC BY-SA 3.0

## 5. The ward data

If we look at the wards table, we have information about the local government of each ward, such as the government office address. This table has 50 rows and 4 columns, or one row for each ward.

## 6. Census data

The census table contains the population of each ward in 2000 and 2010, and that change as a percentage. Additionally, it includes the address for the center of each ward. This table has 50 rows and 6 columns.

## 7. Merging tables

The two tables are related by their ward column. We can merge them together, matching the ward number from each row of the wards table to the ward numbers from the census table. For example, the second ward in the wards table with Alderman Brian Hopkins would be matched with row 2 of the census table where the population in 2000 was 54,361.

## 8. Inner join

The pandas package has an excellent DataFrame method for performing this type of merge called merge. The merge method takes the first DataFrame, wards, and merges it with the second DataFrame, census. We use the on argument to tell the method that we want to merge the two DataFrames on the ward column. Since we listed the wards table first, its columns will appear first in the output, followed by the columns from the census table. In this example, the merge returns a DataFrame with 50 rows and 9 columns, where the returned rows have matching values for the ward column in both tables. This is called an inner join.

## 9. Inner join

An inner join will only return rows that have matching values in both tables.

## 10. Suffixes

You may have noticed that the merged table has columns with suffixes of underscore x or y. This is because both the wards and census tables contained address and zip columns. To avoid multiple columns with the same name, they are automatically given a suffix by the merge method.

## 11. Suffixes

We can use the suffix argument of the merge method to control this behavior. We provide a tuple where all of the overlapping columns in the left table are given the suffix '\_ward', and those of the right table will be given the suffix '\_cen'. This makes it easier for us to tell the difference between the columns.

## 12. Let's practice!

Now let's practice using the merge method.

**Daily XP600**

# Inner join

**50 XP**

## 1. Inner join

Welcome! I am Aaren Stubberfield and I will be your instructor for this course. The pandas package is a powerful tool for manipulating and transforming data in Python. However, when working on an analysis, the data needed could be in multiple tables. This course will focus on the vital skill of merging tables together.

## 2. For clarity

As we start, two quick clarifications. First, through other courses on DataCamp, you may have learned how to import tabular data as DataFrames. In this course, you may hear the words table and DataFrame, but they are equivalent here. Second, we will refer to combining different tables together as merging tables, but note that some refer to this same process as joining.

1. 1 Photo by David Travis on Unsplash

## 3. Chicago data portal dataset

To help us learn about merging tables, we will use data from the city of Chicago data portal.

1. 1 Photo by Pedro Lastra on Unsplash

## 4. Datasets for example

The city of Chicago is divided into fifty local neighborhoods called wards. We have a table with data about the local government offices in each ward. In this example, we want to merge the local government data with census data about the population of each ward.

1. 1 Ward image By Alissapump, Own work, CC BY-SA 3.0

## 5. The ward data

If we look at the wards table, we have information about the local government of each ward, such as the government office address. This table has 50 rows and 4 columns, or one row for each ward.

## 6. Census data

The census table contains the population of each ward in 2000 and 2010, and that change as a percentage. Additionally, it includes the address for the center of each ward. This table has 50 rows and 6 columns.

## 7. Merging tables

The two tables are related by their ward column. We can merge them together, matching the ward number from each row of the wards table to the ward numbers from the census table. For example, the second ward in the wards table with Alderman Brian Hopkins would be matched with row 2 of the census table where the population in 2000 was 54,361.

## 8. Inner join

The pandas package has an excellent DataFrame method for performing this type of merge called merge. The merge method takes the first DataFrame, wards, and merges it with the second DataFrame, census. We use the on argument to tell the method that we want to merge the two DataFrames on the ward column. Since we listed the wards table first, its columns will appear first in the output, followed by the columns from the census table. In this example, the merge returns a DataFrame with 50 rows and 9 columns, where the returned rows have matching values for the ward column in both tables. This is called an inner join.

## 9. Inner join

An inner join will only return rows that have matching values in both tables.

## 10. Suffixes

You may have noticed that the merged table has columns with suffixes of underscore x or y. This is because both the wards and census tables contained address and zip columns. To avoid multiple columns with the same name, they are automatically given a suffix by the merge method.

## 11. Suffixes

We can use the suffix argument of the merge method to control this behavior. We provide a tuple where all of the overlapping columns in the left table are given the suffix '\_ward', and those of the right table will be given the suffix '\_cen'. This makes it easier for us to tell the difference between the columns.

## 12. Let's practice!

Now let's practice using the merge method.

**Daily XP600**

**Exercise**

**Exercise**

**What column to merge on?**

Chicago provides a list of taxicab owners and vehicles licensed to operate within the city, for public safety. Your goal is to merge two tables together. One table is called taxi\_owners, with info about the taxi cab company owners, and one is called taxi\_veh, with info about each taxi cab vehicle. Both the taxi\_owners and taxi\_veh tables have been loaded for you and you can explore them in the IPython shell.

Choose the column you would use to merge the two tables on using the .merge() method.

**Instructions**

**50 XP**

**Possible Answers**

* 

on='rid'

* 

on='vid'

* 

on='year'

* 

on='zip'

In [1]:

print(taxi\_owners)

rid vid owner address zip

0 T6285 6285 AGEAN TAXI LLC 4536 N. ELSTON AVE. 60630

1 T4862 4862 MANGIB CORP. 5717 N. WASHTENAW AVE. 60659

2 T1495 1495 FUNRIDE, INC. 3351 W. ADDISON ST. 60618

3 T4231 4231 ALQUSH CORP. 6611 N. CAMPBELL AVE. 60645

4 T5971 5971 EUNIFFORD INC. 3351 W. ADDISON ST. 60618

... ... ... ... ... ...

3514 T4453 4453 IMAGIN CAB CORP 3351 W. ADDISON ST. 60618

3515 T121 121 TRIBECA CAB CORP 4536 N. ELSTON AVE. 60630

3516 T3465 3465 AMIR EXPRESS INC 3351 W. ADDISON ST. 60618

3517 T1962 1962 KARY CAB COMPANY 4707 N. KENTON AVE. 60630

3518 T1031 1031 NECT 42 LLC 6500 N. WESTERN AVE. 60645

[3519 rows x 5 columns]

In [2]:

print(taxi\_veh)

vid make model year fuel\_type owner

0 2767 TOYOTA CAMRY 2013 HYBRID SEYED M. BADRI

1 1411 TOYOTA RAV4 2017 HYBRID DESZY CORP.

2 6500 NISSAN SENTRA 2019 GASOLINE AGAPH CAB CORP

3 2746 TOYOTA CAMRY 2013 HYBRID MIDWEST CAB CO, INC

4 5922 TOYOTA CAMRY 2013 HYBRID SUMETTI CAB CO

... ... ... ... ... ... ...

3514 5902 TOYOTA CAMRY 2013 HYBRID SAFAR INC

3515 1407 HYUNDAI ELANTRA 2018 GASOLINE MYKONOS CAB CORP.

3516 854 TOYOTA CAMRY 2012 HYBRID JOELIZ CORP INC

3517 6274 TOYOTA CAMRY 2012 HYBRID A K O S INC

3518 4675 FORD ESCAPE 2011 FLEX FUEL MAJAZ CORP

[3519 rows x 6 columns]

**Daily XP650**

**Exercise**

**Exercise**

**Your first inner join**

You have been tasked with figuring out what the most popular types of fuel used in Chicago taxis are. To complete the analysis, you need to merge the taxi\_owners and taxi\_veh tables together on the vid column. You can then use the merged table along with the .value\_counts() method to find the most common fuel\_type.

Since you'll be working with pandas throughout the course, the package will be preloaded for you as pd in each exercise in this course. Also the taxi\_owners and taxi\_veh DataFrames are loaded for you.

**Instructions 1/3**

**35 XP**

* [1](javascript:void(0))
  + Merge taxi\_owners with taxi\_veh on the column vid, and save the result to taxi\_own\_veh.

 [2](javascript:void(0))

* Set the left and right table suffixes for overlapping columns of the merge to \_own and \_veh, respectively.

 [3](javascript:void(0))

* Select the fuel\_type column from taxi\_own\_veh and print the value\_counts() to find the most popular fuel\_types used.
* # Merge the taxi\_owners and taxi\_veh tables
* taxi\_own\_veh = taxi\_owners.\_\_\_\_
* # Print the column names of the taxi\_own\_veh
* print(taxi\_own\_veh.columns)

# Merge the taxi\_owners and taxi\_veh tables setting a suffix

taxi\_own\_veh = taxi\_owners.merge(taxi\_veh, on='vid', \_\_\_\_)

# Print the column names of taxi\_own\_veh

print(taxi\_own\_veh.columns)

# Merge the taxi\_owners and taxi\_veh tables

taxi\_own\_veh = taxi\_owners.merge(taxi\_veh, on='vid')

# Print the column names of the taxi\_own\_veh

print(taxi\_own\_veh.columns)

Index(['rid', 'vid', 'owner\_x', 'address', 'zip', 'make', 'model', 'year', 'fuel\_type', 'owner\_y'], dtype='object')

<script.py> output:

Index(['rid', 'vid', 'owner\_x', 'address', 'zip', 'make', 'model', 'year', 'fuel\_type', 'owner\_y'], dtype='object')

# Merge the taxi\_owners and taxi\_veh tables setting a suffix

taxi\_own\_veh = taxi\_owners.merge(taxi\_veh, on='vid', suffixes= ('\_own','\_veh'))

# Print the column names of taxi\_own\_veh

print(taxi\_own\_veh.columns)

# Merge the taxi\_owners and taxi\_veh tables setting a suffix

taxi\_own\_veh = taxi\_owners.merge(taxi\_veh, on='vid', suffixes= ('\_own','\_veh'))

# Print the column names of taxi\_own\_veh

print(taxi\_own\_veh.columns)

Index(['rid', 'vid', 'owner\_own', 'address', 'zip', 'make', 'model', 'year', 'fuel\_type', 'owner\_veh'], dtype='object')

# Merge the taxi\_owners and taxi\_veh tables setting a suffix

taxi\_own\_veh = taxi\_owners.merge(taxi\_veh, on='vid', suffixes=('\_own','\_veh'))

# Print the value\_counts to find the most popular fuel\_type

print(taxi\_own\_veh['fuel\_type'].value\_counts())

# Merge the taxi\_owners and taxi\_veh tables setting a suffix

taxi\_own\_veh = taxi\_owners.merge(taxi\_veh, on='vid', suffixes= ('\_own','\_veh'))

# Print the column names of taxi\_own\_veh

print(taxi\_own\_veh.columns)

Index(['rid', 'vid', 'owner\_own', 'address', 'zip', 'make', 'model', 'year', 'fuel\_type', 'owner\_veh'], dtype='object')

<script.py> output:

Index(['rid', 'vid', 'owner\_own', 'address', 'zip', 'make', 'model', 'year', 'fuel\_type', 'owner\_veh'], dtype='object')

# Merge the taxi\_owners and taxi\_veh tables setting a suffix

taxi\_own\_veh = taxi\_owners.merge(taxi\_veh, on='vid', suffixes=('\_own','\_veh'))

# Print the value\_counts to find the most popular fuel\_type

print(taxi\_own\_veh['fuel\_type'].value\_counts())

HYBRID 2792

GASOLINE 611

FLEX FUEL 89

COMPRESSED NATURAL GAS 27

Name: fuel\_type, dtype: int64

**Daily XP750**

**Exercise**

**Exercise**

**Inner joins and number of rows returned**

All of the merges you have studied to this point are called inner joins. It is necessary to understand that inner joins only return the rows with matching values in both tables. You will explore this further by reviewing the merge between the wards and census tables, then comparing it to merges of copies of these tables that are slightly altered, named wards\_altered, and census\_altered. The first row of the wards column has been changed in the altered tables. You will examine how this affects the merge between them. The tables have been loaded for you.

For this exercise, it is important to know that the wards and census tables start with **50** rows.

**Instructions 1/3**

**35 XP**

* [1](javascript:void(0))
  + Merge wards and census on the ward column and save the result to wards\_census.

 [2](javascript:void(0))

* Merge the wards\_altered and census tables on the ward column, and notice the difference in returned rows.

 [3](javascript:void(0))

* Merge the wards and census\_altered tables on the ward column, and notice the difference in returned rows.
* # Merge the wards and census tables on the ward column
* wards\_census = wards.merge(\_\_\_\_)
* # Print the shape of wards\_census
* print('wards\_census table shape:', wards\_census.shape)

# Merge the taxi\_owners and taxi\_veh tables setting a suffix taxi\_own\_veh = taxi\_owners.merge(taxi\_veh, on='vid', suffixes=('\_own','\_veh')) # Print the value\_counts to find the most popular fuel\_type print(taxi\_own\_veh['fuel\_type'].value\_counts())

# Print the first few rows of the wards\_altered table to view the change

print(wards\_altered[['ward']].head())

# Merge the wards\_altered and census tables on the ward column

wards\_altered\_census = \_\_\_\_.merge(census, \_\_\_\_)

# Print the shape of wards\_altered\_census

print('wards\_altered\_census table shape:', wards\_altered\_census.shape)

# Merge the wards and census tables on the ward column

wards\_census = wards.merge(census, on='ward')

# Print the shape of wards\_census

print('wards\_census table shape:', wards\_census.shape)

wards\_census table shape: (50, 9)

# Merge the wards and census tables on the ward column

wards\_census = wards.merge(census, on='ward')

print(wards\_census)

# Print the shape of wards\_census

print('wards\_census table shape:', wards\_census.shape)

ward alderman address\_x zip\_x pop\_2000 pop\_2010 change address\_y zip\_y

0 1 Proco "Joe" Moreno 2058 NORTH WESTERN AVENUE 60647 52951 56149 6% 2765 WEST SAINT MARY STREET 60647

1 2 Brian Hopkins 1400 NORTH ASHLAND AVENUE 60622 54361 55805 3% WM WASTE MANAGEMENT 1500 60622

2 3 Pat Dowell 5046 SOUTH STATE STREET 60609 40385 53039 31% 17 EAST 38TH STREET 60653

3 4 William D. Burns 435 EAST 35TH STREET, 1ST FLOOR 60616 51953 54589 5% 31ST ST HARBOR BUILDING LAKEFRONT TRAIL 60653

4 5 Leslie A. Hairston 2325 EAST 71ST STREET 60649 55302 51455 -7% JACKSON PARK LAGOON SOUTH CORNELL DRIVE 60637

5 6 Roderick T. Sawyer 8001 S. MARTIN LUTHER KING DRIVE 60619 54989 52341 -5% 150 WEST 74TH STREET 60636

6 7 Gregory I. Mitchell 2249 EAST 95TH STREET 60617 54593 51581 -6% 8549 SOUTH OGLESBY AVENUE 60617

7 8 Michelle A. Harris 8539 SOUTH COTTAGE GROVE AVENUE 60619 54039 51687 -4% 1346-1352 EAST 75TH STREET 60649

8 9 Anthony A. Beale 34 EAST 112TH PLACE 60628 52008 51519 -1% 11039-11059 SOUTH WENTWORTH AVENUE 60628

9 10 Susan Sadlowski Garza 10500 SOUTH EWING AVENUE 60617 56613 51535 -9% 10534 SOUTH AVENUE F 46394

10 11 Patrick Daley Thompson 3659 SOUTH HALSTED STREET 60609 64228 51497 -20% 943-947 WEST 14TH PLACE 60607

11 12 George Cardenas 3476 SOUTH ARCHER AVENUE 60608 68922 52235 -24% CP 46 STEVENSON EXPRESSWAY 60632

12 13 Marty Quinn 6500 SOUTH PULASKI ROAD 60629 64382 53722 -17% SOUTH RAMP SOUTH LARAMIE AVENUE 60638

13 14 Edward M. Burke 2650 WEST 51ST STREET 60632 80143 54031 -33% 4540 WEST 51ST STREET 60632

14 15 Raymond A. Lopez 1650 WEST 63RD STREET 60636 56057 51501 -8% CHICAGO FIRE DEPARTMENT ENGINE COMPANY 123 2215 60632

15 16 Toni L. Foulkes 3045 WEST 63RD STREET 60629 50205 51954 3% 6036 SOUTH WOOD STREET 60636

16 17 David H. Moore 7313 SOUTH ASHLAND AVENUE 60636 49264 51846 5% 7216 SOUTH WINCHESTER AVENUE 60636

17 18 Derrick G. Curtis 8359 SOUTH PULASKI ROAD 60652 55043 52992 -4% 3286 WEST COLUMBUS AVENUE 60652

18 19 Matthew J. O'Shea 10400 SOUTH WESTERN AVENUE 60643 54546 51525 -6% 9999 SOUTH FRANCISCO AVENUE 60805

19 20 Willie B. Cochran 6357 SOUTH COTTAGE GROVE AVENUE 60637 51854 52372 1% DAN RYAN EXPRESSWAY PARK MANOR 60621

20 21 Howard B. Brookins, Jr. 9011 SOUTH ASHLAND AVENUE, UNIT B 60620 51751 51632 0% 8852-8854 SOUTH EMERALD AVENUE 60620

21 22 Ricardo Munoz 2500 SOUTH ST. LOUIS AVENUE 60623 59734 53515 -10% 4233 WEST 36TH STREET 60632

22 23 Michael R. Zalewski 6247 SOUTH ARCHER AVENUE 60638 63691 53728 -16% CHICAGO MIDWAY INTERNATIONAL AIRPORT WEST 62ND... 60629

23 24 Michael Scott, Jr. 1158 SOUTH KEELER AVENUE 60624 50879 54909 8% 1635 SOUTH CHRISTIANA AVENUE 60623

24 25 Daniel "Danny" Solis 1800 SOUTH BLUE ISLAND AVENUE 60608 55954 54539 -3% 1632-1746 SOUTH MILLER STREET 60608

25 26 Roberto Maldonado 2511 WEST DIVISION STREET 60622 56841 53516 -6% LITTLE CUBS FIELD COMFORT STATION 1400 60622

26 27 Walter Burnett, Jr. 4 NORTH WESTERN AVENUE 60612 61287 52939 -14% 2151-2153 WEST CHICAGO AVENUE 60651

27 28 Jason C. Ervin 2602 WEST 16TH STREET 60612 49423 55199 12% RML SPECIALTY HOSPITAL 3435 60624

28 29 Chris Taliaferro 6272 WEST NORTH AVENUE 60639 61949 55267 -11% 1241 NORTH RIDGELAND AVENUE 60302

29 30 Ariel E. Reyboras 3559 NORTH MILWAUKEE AVENUE 60641 72698 55560 -24% 5118 WEST FLETCHER STREET 60641

30 31 Milagros "Milly" Santiago 2521 NORTH PULASKI ROAD 60639 65045 53724 -17% 2854 NORTH KEATING AVENUE 60641

31 32 Scott Waguespack 2657 NORTH CLYBOURN AVENUE 60614 57204 55184 -4% 2901 NORTH WASHTENAW AVENUE 60618

32 33 Deborah Mell 3001 WEST IRVING PARK ROAD 60618 63695 55598 -13% 4041-4043 NORTH RICHMOND STREET 60625

33 34 Carrie M. Austin 507 WEST 111TH STREET 60628 49922 51599 3% 11544-11546 SOUTH PEORIA STREET 60827

34 35 Carlos Ramirez-Rosa 2710 NORTH SAWYER AVENUE 60647 57588 55281 -4% 3634 WEST BELMONT AVENUE 60618

35 36 Gilbert Villegas 6934 WEST DIVERSEY 60607 63376 54766 -14% 2918 NORTH RUTHERFORD AVENUE 60634

36 37 Emma M. Mitts 4924 WEST CHICAGO AVENUE 60651 56120 51538 -8% 4738-4748 WEST RICE STREET 60651

37 38 Nicholas Sposato 3821 NORTH HARLEM AVENUE 60634 66011 56001 -15% 7307-7331 WEST IRVING PARK ROAD 60706

38 39 Margaret Laurino 4404 WEST LAWRENCE AVENUE 60630 64291 55882 -13% QUEEN OF ALL SAINTS BASILICA 6280 60646

39 40 Patrick J. O'Connor 5850 NORTH LINCOLN AVENUE 60659 58652 55319 -6% 5536 NORTH ARTESIAN AVENUE 60645

40 41 Anthony V. Napolitano 7442 NORTH HARLEM AVENUE 60631 56127 55991 0% 1652 SOUTH CLIFTON AVENUE 60068

41 42 Brendan Reilly 325 WEST HURON STREET, SUITE 510 60654 68102 55870 -18% 410-420 WEST GRAND AVENUE 60654

42 43 Michelle Smith 2523 NORTH HALSTED STREET 60614 57668 56170 -3% LINCOLN PARK ZOO 2001 60614

43 44 Tom Tunney 3223 NORTH SHEFFIELD AVENUE 60657 58758 56058 -5% 507-513 WEST ALDINE AVENUE 60657

44 45 John S. Arena 4754 NORTH MILWAUKEE AVENUE 60630 60653 55967 -8% CONGREGATIONAL CHURCH OF JEFFERSON PARK 5320 60630

45 46 James Cappleman 4544 NORTH BROADWAY AVENUE 60640 56587 53784 -5% UPTOWN BROADWAY BUILDING 4743-4763 60640

46 47 Ameya Pawar 4243 NORTH LINCOLN AVENUE 60618 52108 55074 6% 2153 WEST BERTEAU AVENUE 60618

47 48 Harry Osterman 5533 NORTH BROADWAY AVENUE 60640 56246 55014 -2% 1025 WEST HOLLYWOOD AVENUE 60660

48 49 Joe Moore 7356 NORTH GREENVIEW AVENUE 60626 59435 54633 -8% 1426 WEST ESTES AVENUE 60645

49 50 Debra L. Silverstein 2949 WEST DEVON AVENUE, SUITE A 60659 62383 55809 -11% 2638 WEST NORTH SHORE AVENUE 60645

wards\_census table shape: (50, 9)

<script.py> output:

ward alderman address\_x zip\_x pop\_2000 pop\_2010 change address\_y zip\_y

0 1 Proco "Joe" Moreno 2058 NORTH WESTERN AVENUE 60647 52951 56149 6% 2765 WEST SAINT MARY STREET 60647

1 2 Brian Hopkins 1400 NORTH ASHLAND AVENUE 60622 54361 55805 3% WM WASTE MANAGEMENT 1500 60622

2 3 Pat Dowell 5046 SOUTH STATE STREET 60609 40385 53039 31% 17 EAST 38TH STREET 60653

3 4 William D. Burns 435 EAST 35TH STREET, 1ST FLOOR 60616 51953 54589 5% 31ST ST HARBOR BUILDING LAKEFRONT TRAIL 60653

4 5 Leslie A. Hairston 2325 EAST 71ST STREET 60649 55302 51455 -7% JACKSON PARK LAGOON SOUTH CORNELL DRIVE 60637

5 6 Roderick T. Sawyer 8001 S. MARTIN LUTHER KING DRIVE 60619 54989 52341 -5% 150 WEST 74TH STREET 60636

6 7 Gregory I. Mitchell 2249 EAST 95TH STREET 60617 54593 51581 -6% 8549 SOUTH OGLESBY AVENUE 60617

7 8 Michelle A. Harris 8539 SOUTH COTTAGE GROVE AVENUE 60619 54039 51687 -4% 1346-1352 EAST 75TH STREET 60649

8 9 Anthony A. Beale 34 EAST 112TH PLACE 60628 52008 51519 -1% 11039-11059 SOUTH WENTWORTH AVENUE 60628

9 10 Susan Sadlowski Garza 10500 SOUTH EWING AVENUE 60617 56613 51535 -9% 10534 SOUTH AVENUE F 46394

10 11 Patrick Daley Thompson 3659 SOUTH HALSTED STREET 60609 64228 51497 -20% 943-947 WEST 14TH PLACE 60607

11 12 George Cardenas 3476 SOUTH ARCHER AVENUE 60608 68922 52235 -24% CP 46 STEVENSON EXPRESSWAY 60632

12 13 Marty Quinn 6500 SOUTH PULASKI ROAD 60629 64382 53722 -17% SOUTH RAMP SOUTH LARAMIE AVENUE 60638

13 14 Edward M. Burke 2650 WEST 51ST STREET 60632 80143 54031 -33% 4540 WEST 51ST STREET 60632

14 15 Raymond A. Lopez 1650 WEST 63RD STREET 60636 56057 51501 -8% CHICAGO FIRE DEPARTMENT ENGINE COMPANY 123 2215 60632

15 16 Toni L. Foulkes 3045 WEST 63RD STREET 60629 50205 51954 3% 6036 SOUTH WOOD STREET 60636

16 17 David H. Moore 7313 SOUTH ASHLAND AVENUE 60636 49264 51846 5% 7216 SOUTH WINCHESTER AVENUE 60636

17 18 Derrick G. Curtis 8359 SOUTH PULASKI ROAD 60652 55043 52992 -4% 3286 WEST COLUMBUS AVENUE 60652

18 19 Matthew J. O'Shea 10400 SOUTH WESTERN AVENUE 60643 54546 51525 -6% 9999 SOUTH FRANCISCO AVENUE 60805

19 20 Willie B. Cochran 6357 SOUTH COTTAGE GROVE AVENUE 60637 51854 52372 1% DAN RYAN EXPRESSWAY PARK MANOR 60621

20 21 Howard B. Brookins, Jr. 9011 SOUTH ASHLAND AVENUE, UNIT B 60620 51751 51632 0% 8852-8854 SOUTH EMERALD AVENUE 60620

21 22 Ricardo Munoz 2500 SOUTH ST. LOUIS AVENUE 60623 59734 53515 -10% 4233 WEST 36TH STREET 60632

22 23 Michael R. Zalewski 6247 SOUTH ARCHER AVENUE 60638 63691 53728 -16% CHICAGO MIDWAY INTERNATIONAL AIRPORT WEST 62ND... 60629

23 24 Michael Scott, Jr. 1158 SOUTH KEELER AVENUE 60624 50879 54909 8% 1635 SOUTH CHRISTIANA AVENUE 60623

24 25 Daniel "Danny" Solis 1800 SOUTH BLUE ISLAND AVENUE 60608 55954 54539 -3% 1632-1746 SOUTH MILLER STREET 60608

25 26 Roberto Maldonado 2511 WEST DIVISION STREET 60622 56841 53516 -6% LITTLE CUBS FIELD COMFORT STATION 1400 60622

26 27 Walter Burnett, Jr. 4 NORTH WESTERN AVENUE 60612 61287 52939 -14% 2151-2153 WEST CHICAGO AVENUE 60651

27 28 Jason C. Ervin 2602 WEST 16TH STREET 60612 49423 55199 12% RML SPECIALTY HOSPITAL 3435 60624

28 29 Chris Taliaferro 6272 WEST NORTH AVENUE 60639 61949 55267 -11% 1241 NORTH RIDGELAND AVENUE 60302

29 30 Ariel E. Reyboras 3559 NORTH MILWAUKEE AVENUE 60641 72698 55560 -24% 5118 WEST FLETCHER STREET 60641

30 31 Milagros "Milly" Santiago 2521 NORTH PULASKI ROAD 60639 65045 53724 -17% 2854 NORTH KEATING AVENUE 60641

31 32 Scott Waguespack 2657 NORTH CLYBOURN AVENUE 60614 57204 55184 -4% 2901 NORTH WASHTENAW AVENUE 60618

32 33 Deborah Mell 3001 WEST IRVING PARK ROAD 60618 63695 55598 -13% 4041-4043 NORTH RICHMOND STREET 60625

33 34 Carrie M. Austin 507 WEST 111TH STREET 60628 49922 51599 3% 11544-11546 SOUTH PEORIA STREET 60827

34 35 Carlos Ramirez-Rosa 2710 NORTH SAWYER AVENUE 60647 57588 55281 -4% 3634 WEST BELMONT AVENUE 60618

35 36 Gilbert Villegas 6934 WEST DIVERSEY 60607 63376 54766 -14% 2918 NORTH RUTHERFORD AVENUE 60634

36 37 Emma M. Mitts 4924 WEST CHICAGO AVENUE 60651 56120 51538 -8% 4738-4748 WEST RICE STREET 60651

37 38 Nicholas Sposato 3821 NORTH HARLEM AVENUE 60634 66011 56001 -15% 7307-7331 WEST IRVING PARK ROAD 60706

38 39 Margaret Laurino 4404 WEST LAWRENCE AVENUE 60630 64291 55882 -13% QUEEN OF ALL SAINTS BASILICA 6280 60646

39 40 Patrick J. O'Connor 5850 NORTH LINCOLN AVENUE 60659 58652 55319 -6% 5536 NORTH ARTESIAN AVENUE 60645

40 41 Anthony V. Napolitano 7442 NORTH HARLEM AVENUE 60631 56127 55991 0% 1652 SOUTH CLIFTON AVENUE 60068

41 42 Brendan Reilly 325 WEST HURON STREET, SUITE 510 60654 68102 55870 -18% 410-420 WEST GRAND AVENUE 60654

42 43 Michelle Smith 2523 NORTH HALSTED STREET 60614 57668 56170 -3% LINCOLN PARK ZOO 2001 60614

43 44 Tom Tunney 3223 NORTH SHEFFIELD AVENUE 60657 58758 56058 -5% 507-513 WEST ALDINE AVENUE 60657

44 45 John S. Arena 4754 NORTH MILWAUKEE AVENUE 60630 60653 55967 -8% CONGREGATIONAL CHURCH OF JEFFERSON PARK 5320 60630

45 46 James Cappleman 4544 NORTH BROADWAY AVENUE 60640 56587 53784 -5% UPTOWN BROADWAY BUILDING 4743-4763 60640

46 47 Ameya Pawar 4243 NORTH LINCOLN AVENUE 60618 52108 55074 6% 2153 WEST BERTEAU AVENUE 60618

47 48 Harry Osterman 5533 NORTH BROADWAY AVENUE 60640 56246 55014 -2% 1025 WEST HOLLYWOOD AVENUE 60660

48 49 Joe Moore 7356 NORTH GREENVIEW AVENUE 60626 59435 54633 -8% 1426 WEST ESTES AVENUE 60645

49 50 Debra L. Silverstein 2949 WEST DEVON AVENUE, SUITE A 60659 62383 55809 -11% 2638 WEST NORTH SHORE AVENUE 60645

wards\_census table shape: (50, 9)

# Print the first few rows of the wards\_altered table to view the change

print(wards\_altered[['ward']].head())

# Merge the wards\_altered and census tables on the ward column

wards\_altered\_census = wards\_altered.merge(census, on='ward')

# Print the shape of wards\_altered\_census

print('wards\_altered\_census table shape:', wards\_altered\_census.shape)

# Print the first few rows of the wards\_altered table to view the change

print(wards\_altered[['ward']].head())

# Merge the wards\_altered and census tables on the ward column

wards\_altered\_census = wards\_altered.merge(census, on='ward')

# Print the shape of wards\_altered\_census

print('wards\_altered\_census table shape:', wards\_altered\_census.shape)

ward

0 61

1 2

2 3

3 4

4 5

wards\_altered\_census table shape: (49, 9)

**Daily XP850**

# One-to-many relationships

**50 XP**

## 1. One to many relationships

Welcome back! In the last lesson, we learned how to merge two DataFrames together with the merge method. In this lesson, we'll discuss different types of relationships between tables. In particular, we will discuss the one-to-many relationship. But first, let's quickly consider what a one-to-one relationship is.

## 2. One-to-one

In a one-to-one relationship, every row in the left table is related to one and only one row in the right table.

## 3. One-to-one example

We looked at a one-to-one relationship earlier. Recall the relationship between the wards table and the census table. Every row in the wards table is related to only one row in the census table, so there is only one row for ward 3 in each table. Practically speaking, it only makes sense that there is one row of population information for each ward. It wouldn't make sense if the census table contained multiple population values in 2000 for the third ward.

## 4. One-to-many

So, what is a one-to-many relationship? Well, in a one-to-many relationship, every row in the left table is related to one or more rows in the right table.

## 5. One-to-many example

To provide an example of a one-to-many relationship, let's think back to our wards table. Within each ward, there are many businesses. We will merge the wards table with a table of licensed businesses in each ward.

## 6. One-to-many example

The business license data is stored in another table called licenses. It holds info such as the business address and ward the business is located within.

## 7. One-to-many example

The two DataFrames are related to each other by their ward column.

## 8. One-to-many example

When we merge the two tables together with the merge method, setting the 'on' attribute to the column ward, the resulting table has both local ward data and business license data. Notice that ward 1 and its alderman Joe is repeated in the resulting table because the licenses table has many businesses in the 1st ward. pandas takes care of the one-to-many relationships for us and doesn't require anything special on our end. We can use the same syntax as we did with one-to-one relationships.

## 9. One-to-many example

By printing the shape, we can see that our original wards table has 50 rows. After merging the wards table with the licenses table, the resulting table has 10,000 rows. When you merge tables that have a one-to-many relationship, the number of rows returned will likely be different than the number in the left table.

## 10. Let's practice!

Now let's make the one-to-many relationship idea more concrete by practicing.

**Daily XP900**

**Exercise**

**One-to-many classification**

Understanding the difference between a one-to-one and one-to-many relationship is a useful skill. In this exercise, consider a set of tables from an e-commerce website. The hypothetical tables are the following:

* A customer table with information about each customer
* A cust\_tax\_info table with customers unique tax IDs
* An orders table with information about each order
* A products table with details about each unique product sold
* An inventory table with information on how much total inventory is available to sell for each product

**Instructions**

**100XP**

* Select the relationship type that is most appropriate for the relationship between the different tables: **One-to-one**, or **One-to-many**.

**Daily XP1000**

**Exercise**

**Exercise**

**One-to-many merge**

A business may have one or multiple owners. In this exercise, you will continue to gain experience with one-to-many merges by merging a table of business owners, called biz\_owners, to the licenses table. Recall from the video lesson, with a one-to-many relationship, a row in the left table may be repeated if it is related to multiple rows in the right table. In this lesson, you will explore this further by finding out what is the most common business owner title. (i.e., secretary, CEO, or vice president)

The licenses and biz\_owners DataFrames are loaded for you.

**Instructions**

**100 XP**

* Starting with the licenses table on the left, merge it to the biz\_owners table on the column account, and save the results to a variable named licenses\_owners.
* Group licenses\_owners by title and count the number of accounts for each title. Save the result as counted\_df
* Sort counted\_df by the number of **accounts** in **descending order**, and save this as a variable named sorted\_df.
* Use the .head() method to print the first few rows of the sorted\_df.
* # Merge the licenses and biz\_owners table on account
* licenses\_owners = \_\_\_\_
* # Group the results by title then count the number of accounts
* counted\_df = licenses\_owners.groupby(\_\_\_\_).agg({'account':'count'})
* # Sort the counted\_df in desending order
* sorted\_df = counted\_df.sort\_values(\_\_\_\_)
* # Use .head() method to print the first few rows of sorted\_df
* print(\_\_\_\_)

# Print the first few rows of the census\_altered table to view the change print(census\_altered[['ward']].head()) # Merge the wards and census\_altered tables on the ward column wards\_census\_altered = wards.merge(census\_altered, on='ward') # Print the shape of wards\_census\_altered print('wards\_census\_altered table shape:', wards\_census\_altered.shape)

# 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('account',ascending=False)

# Use .head() method to print the first few rows of sorted\_df

print(sorted\_df.head())

# 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('account',ascending=False)

# Use .head() method to print the first few rows of sorted\_df

print(sorted\_df.head())

account

title

PRESIDENT 6259

SECRETARY 5205

SOLE PROPRIETOR 1658

OTHER 1200

VICE PRESIDENT 970

**Daily XP1100**

# Merging multiple DataFrames

**50 XP**

## 1. Merging multiple DataFrames

Welcome back. In our last lesson, we learned how to merge two tables with a one-to-many relationship using the merge method. Merging data like this is a necessary skill to bring together data from different sources to answer some more complex data questions.

## 2. Merging multiple tables

Sometimes we need to merge together more than just two tables to complete our analysis.

## 3. Remembering the licenses table

In the previous lesson, we used two tables from the city of Chicago. One table contained business licenses issued by the city.

## 4. Remembering the wards table

The other table listed info about the local neighborhoods called wards, including the local government official's office.

## 5. Review new data

Now, we also have a table of businesses that have received small business grant money from Chicago. The grants are funded by taxpayer money. Therefore, it would be helpful to analyze how much grant money each business received and in what ward that business is located. We then could determine if one ward's businesses received a disproportionately large amount of grant money.

## 6. Tables to merge

To pull all of this information together, let's first connect our grants table to our licenses table. The two tables are related by their company name and location. Let's pause here for a moment.

## 7. Theoretical merge

If we merge the two tables only using the zip column, then the 60616 zip of Reggie's bar from the licenses table will be matched to multiple businesses in the grants table with the same zip. Our code sample prints the first few rows and some columns of the merged table. The output of the merge duplicates Reggie's bar for each matching zip in the grants table, which is not what we want. If instead, we merged on address only, there's a small risk that the address would repeat in different parts of the city. Therefore, the best option is to merge the tables using the combination of both address and zip code.

## 8. Single merge

We merge the two DataFrames as shown before, except in this case, we pass a list of the column names we want to merge on to the 'on' argument. This allows us to use multiple columns in the merge. As before, the matching rows between the two DataFrames are returned with the columns from the grant table listed first. However, when we merge on two columns, in this case address and zip code, we are requiring that both the address and zip code of a row in the left table match the address and zip code of a row in the right table in order for them to be linked to each other in the merge.

## 9. Merging multiple tables

We can now extend this example to a third table. First, we merge the grants table with the wards table on the ward column again, adding suffixes to the repeated column names. Note that we're using Python's backslash line continuation method to add the second merge on the next line. Python will read this as just one line of code. Without this, Python will throw a syntax error since it will parse it as two separate lines of code, so don't forget your backslash. Now our output table has information about grants, business, and wards. We can now complete our analysis.

## 10. Results

We can now sum the grants by ward and plot the results. Some wards have received more grants than others.

## 11. Merging even more...

We could continue to merge additional tables as needed. We stopped at three, but if needed, we could continue to add more. The code here shows the pattern you would follow as you merge more tables.

## 12. Let's practice!

Now, let's practice merging multiple tables.

**Daily XP1150**

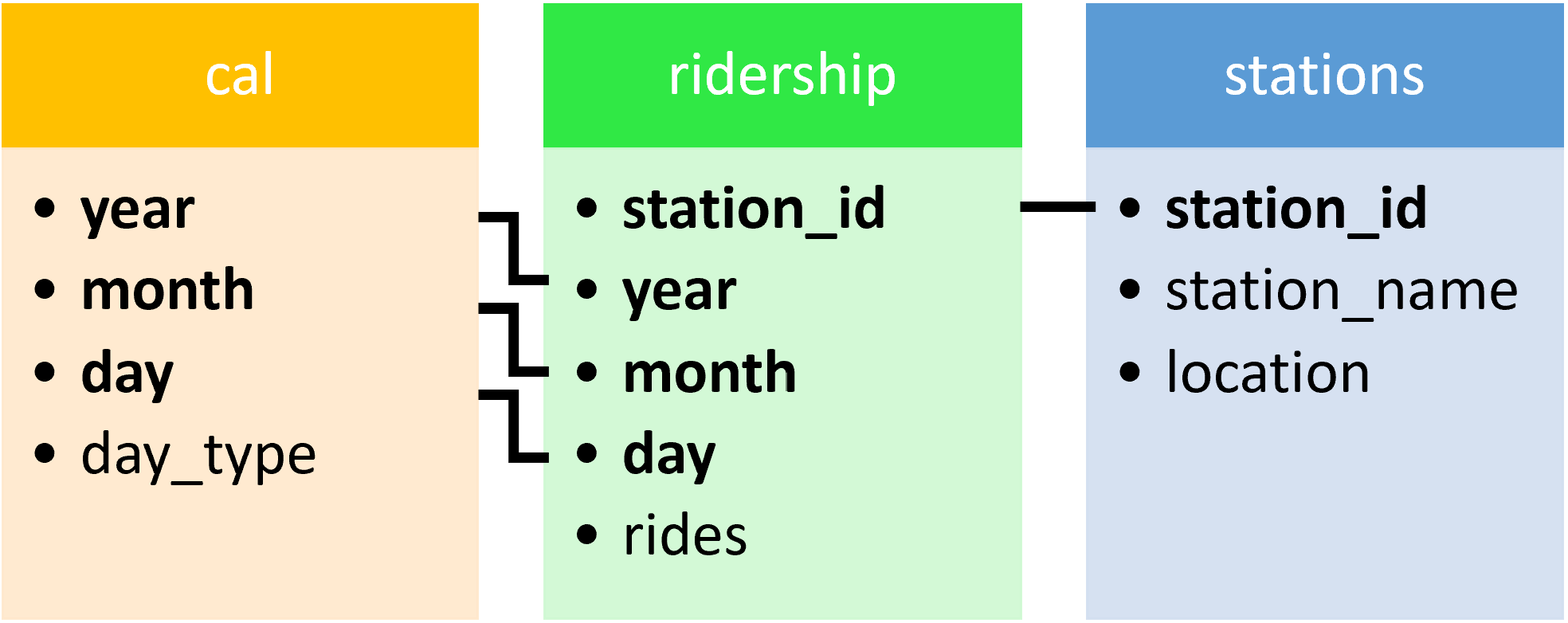
**Exercise**

**Exercise**

**Total riders in a month**

Your goal is to find the total number of rides provided to passengers passing through the Wilson station (station\_name == 'Wilson') when riding Chicago's public transportation system on weekdays (day\_type == 'Weekday') in July (month == 7). Luckily, Chicago provides this detailed data, but it is in three different tables. You will work on merging these tables together to answer the question. This data is different from the business related data you have seen so far, but all the information you need to answer the question is provided.

The cal, ridership, and stations DataFrames have been loaded for you. The relationship between the tables can be seen in the diagram below.



**Instructions 1/3**

**35 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))
* Merge the ridership and cal tables together, starting with the ridership table on the left and save the result to the variable ridership\_cal. If you code takes too long to run, your merge conditions might be incorrect.
* # Merge the ridership and cal tables
* ridership\_cal = ridership.merge(\_\_\_\_)

# 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('account',ascending=False) # Use .head() method to print the first few rows of sorted\_df print(sorted\_df.head())

# Merge the ridership and cal tables

ridership\_cal = ridership.merge(cal, on=['year', 'month', 'day'])

# Merge the ridership and cal tables

ridership\_cal = ridership.merge(cal, on=['year', 'month', 'day'])

# Merge the ridership and cal tables

ridership\_cal = ridership.merge(cal, on=['year', 'month', 'day'])

print(ridership\_cal)

station\_id year month day rides day\_type

0 40010 2019 1 1 576 Sunday/Holiday

1 40080 2019 1 1 1839 Sunday/Holiday

2 40770 2019 1 1 2724 Sunday/Holiday

3 40120 2019 1 1 754 Sunday/Holiday

4 40540 2019 1 1 2175 Sunday/Holiday

... ... ... ... ... ... ...

3280 40540 2019 12 31 4355 Weekday

3281 41260 2019 12 31 1228 Weekday

3282 41500 2019 12 31 1685 Weekday

3283 41440 2019 12 31 1370 Weekday

3284 41660 2019 12 31 13430 Weekday

[3285 rows x 6 columns]

<script.py> output:

station\_id year month day rides day\_type

0 40010 2019 1 1 576 Sunday/Holiday

1 40080 2019 1 1 1839 Sunday/Holiday

2 40770 2019 1 1 2724 Sunday/Holiday

3 40120 2019 1 1 754 Sunday/Holiday

4 40540 2019 1 1 2175 Sunday/Holiday

... ... ... ... ... ... ...

3280 40540 2019 12 31 4355 Weekday

3281 41260 2019 12 31 1228 Weekday

3282 41500 2019 12 31 1685 Weekday

3283 41440 2019 12 31 1370 Weekday

3284 41660 2019 12 31 13430 Weekday

[3285 rows x 6 columns]

# Merge the ridership, cal, and stations tables

ridership\_cal\_stations = ridership.merge(cal, on=['year','month','day']) \

.merge(stations, on='station\_id')

print(ridership\_cal\_stations)

station\_id year month day rides day\_type station\_name location

0 40010 2019 1 1 576 Sunday/Holiday Austin-Forest Park (41.870851, -87.776812)

1 40010 2019 1 2 1457 Weekday Austin-Forest Park (41.870851, -87.776812)

2 40010 2019 1 3 1543 Weekday Austin-Forest Park (41.870851, -87.776812)

3 40010 2019 1 4 1621 Weekday Austin-Forest Park (41.870851, -87.776812)

4 40010 2019 1 5 719 Saturday Austin-Forest Park (41.870851, -87.776812)

... ... ... ... ... ... ... ... ...

3280 41660 2019 12 27 13898 Weekday Lake/State (41.884809, -87.627813)

3281 41660 2019 12 28 9485 Saturday Lake/State (41.884809, -87.627813)

3282 41660 2019 12 29 7581 Sunday/Holiday Lake/State (41.884809, -87.627813)

3283 41660 2019 12 30 15332 Weekday Lake/State (41.884809, -87.627813)

3284 41660 2019 12 31 13430 Weekday Lake/State (41.884809, -87.627813)

Create a variable called filter\_criteria to select the appropriate rows from the merged table so that you can sum the rides column.

# Merge the ridership, cal, and stations tables

ridership\_cal\_stations = ridership.merge(cal, on=['year','month','day']) \

                            .merge(stations, on='station\_id')

# Create a filter to filter ridership\_cal\_stations

filter\_criteria = ((ridership\_cal\_stations['month'] == 7)

                   & (ridership\_cal\_stations['day\_type'] == 'Weekday')

                   & (ridership\_cal\_stations['station\_name'] == 'Wilson'))

# Use .loc and the filter to select for rides

print(ridership\_cal\_stations.loc[filter\_criteria, 'rides'].sum())

ridership\_cal = ridership.merge(cal, on=['year', 'month', 'day'])

# Merge the ridership and cal tables

ridership\_cal = ridership.merge(cal, on=['year', 'month', 'day'])

# Merge the ridership and cal tables

ridership\_cal = ridership.merge(cal, on=['year', 'month', 'day'])

print(ridership\_cal)

station\_id year month day rides day\_type

0 40010 2019 1 1 576 Sunday/Holiday

1 40080 2019 1 1 1839 Sunday/Holiday

2 40770 2019 1 1 2724 Sunday/Holiday

3 40120 2019 1 1 754 Sunday/Holiday

4 40540 2019 1 1 2175 Sunday/Holiday

... ... ... ... ... ... ...

3280 40540 2019 12 31 4355 Weekday

3281 41260 2019 12 31 1228 Weekday

3282 41500 2019 12 31 1685 Weekday

3283 41440 2019 12 31 1370 Weekday

3284 41660 2019 12 31 13430 Weekday

[3285 rows x 6 columns]

<script.py> output:

station\_id year month day rides day\_type

0 40010 2019 1 1 576 Sunday/Holiday

1 40080 2019 1 1 1839 Sunday/Holiday

2 40770 2019 1 1 2724 Sunday/Holiday

3 40120 2019 1 1 754 Sunday/Holiday

4 40540 2019 1 1 2175 Sunday/Holiday

... ... ... ... ... ... ...

3280 40540 2019 12 31 4355 Weekday

3281 41260 2019 12 31 1228 Weekday

3282 41500 2019 12 31 1685 Weekday

3283 41440 2019 12 31 1370 Weekday

3284 41660 2019 12 31 13430 Weekday

[3285 rows x 6 columns]

# Merge the ridership, cal, and stations tables

ridership\_cal\_stations = ridership.merge(cal, on=['year','month','day']) \

.merge(stations, on='station\_id')

print(ridership\_cal\_stations)

station\_id year month day rides day\_type station\_name location

0 40010 2019 1 1 576 Sunday/Holiday Austin-Forest Park (41.870851, -87.776812)

1 40010 2019 1 2 1457 Weekday Austin-Forest Park (41.870851, -87.776812)

2 40010 2019 1 3 1543 Weekday Austin-Forest Park (41.870851, -87.776812)

3 40010 2019 1 4 1621 Weekday Austin-Forest Park (41.870851, -87.776812)

4 40010 2019 1 5 719 Saturday Austin-Forest Park (41.870851, -87.776812)

... ... ... ... ... ... ... ... ...

3280 41660 2019 12 27 13898 Weekday Lake/State (41.884809, -87.627813)

3281 41660 2019 12 28 9485 Saturday Lake/State (41.884809, -87.627813)

3282 41660 2019 12 29 7581 Sunday/Holiday Lake/State (41.884809, -87.627813)

3283 41660 2019 12 30 15332 Weekday Lake/State (41.884809, -87.627813)

3284 41660 2019 12 31 13430 Weekday Lake/State (41.884809, -87.627813)

[3285 rows x 8 columns]

<script.py> output:

station\_id year month day rides day\_type station\_name location

0 40010 2019 1 1 576 Sunday/Holiday Austin-Forest Park (41.870851, -87.776812)

1 40010 2019 1 2 1457 Weekday Austin-Forest Park (41.870851, -87.776812)

2 40010 2019 1 3 1543 Weekday Austin-Forest Park (41.870851, -87.776812)

3 40010 2019 1 4 1621 Weekday Austin-Forest Park (41.870851, -87.776812)

4 40010 2019 1 5 719 Saturday Austin-Forest Park (41.870851, -87.776812)

... ... ... ... ... ... ... ... ...

3280 41660 2019 12 27 13898 Weekday Lake/State (41.884809, -87.627813)

3281 41660 2019 12 28 9485 Saturday Lake/State (41.884809, -87.627813)

3282 41660 2019 12 29 7581 Sunday/Holiday Lake/State (41.884809, -87.627813)

3283 41660 2019 12 30 15332 Weekday Lake/State (41.884809, -87.627813)

3284 41660 2019 12 31 13430 Weekday Lake/State (41.884809, -87.627813)

[3285 rows x 8 columns]

# Merge the ridership, cal, and stations tables

ridership\_cal\_stations = ridership.merge(cal, on=['year','month','day']) \

.merge(stations, on='station\_id')

# Create a filter to filter ridership\_cal\_stations

filter\_criteria = ((ridership\_cal\_stations['month'] == 7)

& (ridership\_cal\_stations['day\_type'] == 'Weekday')

& (ridership\_cal\_stations['station\_name'] == 'Wilson'))

# Use .loc and the filter to select for rides

print(ridership\_cal\_stations.loc[filter\_criteria, 'rides'].sum())

140005

**Daily XP1250**

**Exercise**

**Exercise**

**Three table merge**

To solidify the concept of a three DataFrame merge, practice another exercise. A reasonable extension of our review of Chicago business data would include looking at demographics information about the neighborhoods where the businesses are. A table with the median income by zip code has been provided to you. You will merge the licenses and wards tables with this new income-by-zip-code table called zip\_demo.

The licenses, wards, and zip\_demo DataFrames have been loaded for you.

**Instructions**

**100 XP**

* Starting with the licenses table, merge to it the zip\_demo table on the zip column. Then merge the resulting table to the wards table on the ward column. Save result of the three merged tables to a variable named licenses\_zip\_ward.
* Group the results of the three merged tables by the column alderman and find the median income.
* # Merge licenses and zip\_demo, on zip; and merge the wards on ward
* licenses\_zip\_ward = licenses.merge\_\_\_\_ \
* \_\_\_\_
* # Print the results by alderman and show median income
* print(\_\_\_\_.groupby(\_\_\_\_).agg({'income':'median'}))

# Merge the ridership, cal, and stations tables ridership\_cal\_stations = ridership.merge(cal, on=['year','month','day']) \ .merge(stations, on='station\_id') # Create a filter to filter ridership\_cal\_stations filter\_criteria = ((ridership\_cal\_stations['month'] == 7) & (ridership\_cal\_stations['day\_type'] == 'Weekday') & (ridership\_cal\_stations['station\_name'] == 'Wilson')) # Use .loc and the filter to select for rides print(ridership\_cal\_stations.loc[filter\_criteria, 'rides'].sum())

# Merge licenses and zip\_demo, on zip; and merge the wards on ward

licenses\_zip\_ward = licenses.merge(zip\_demo, on= 'zip') \

                        .merge(wards, on= 'ward')

# Print the results by alderman and show median income

print(licenses\_zip\_ward.groupby('alderman').agg({'income':'median'}))

# Merge licenses and zip\_demo, on zip; and merge the wards on ward

licenses\_zip\_ward = licenses.merge(zip\_demo, on= 'zip') \

.merge(wards, on= 'ward')

# Print the results by alderman and show median income

print(licenses\_zip\_ward.groupby('alderman').agg({'income':'median'}))

income

alderman

Ameya Pawar 66246.0

Anthony A. Beale 38206.0

Anthony V. Napolitano 82226.0

Ariel E. Reyboras 41307.0

Brendan Reilly 110215.0

Brian Hopkins 87143.0

Carlos Ramirez-Rosa 66246.0

Carrie M. Austin 38206.0

Chris Taliaferro 55566.0

Daniel "Danny" Solis 41226.0

David H. Moore 33304.0

Deborah Mell 66246.0

Debra L. Silverstein 50554.0

Derrick G. Curtis 65770.0

Edward M. Burke 42335.0

Emma M. Mitts 36283.0

George Cardenas 33959.0

Gilbert Villegas 41307.0

Gregory I. Mitchell 24941.0

Harry Osterman 45442.0

Howard B. Brookins, Jr. 33304.0

James Cappleman 79565.0

Jason C. Ervin 41226.0

Joe Moore 39163.0

John S. Arena 70122.0

Leslie A. Hairston 28024.0

Margaret Laurino 70122.0

Marty Quinn 67045.0

Matthew J. O'Shea 59488.0

Michael R. Zalewski 42335.0

Michael Scott, Jr. 31445.0

Michelle A. Harris 32558.0

Michelle Smith 100116.0

Milagros "Milly" Santiago 41307.0

Nicholas Sposato 62223.0

Pat Dowell 46340.0

Patrick Daley Thompson 41226.0

Patrick J. O'Connor 50554.0

Proco "Joe" Moreno 87143.0

Raymond A. Lopez 33959.0

Ricardo Munoz 31445.0

Roberto Maldonado 68223.0

Roderick T. Sawyer 32558.0

Scott Waguespack 68223.0

Susan Sadlowski Garza 38417.0

Tom Tunney 88708.0

Toni L. Foulkes 27573.0

Walter Burnett, Jr. 87143.0

William D. Burns 107811.0

Willie B. Cochran 28024.0

**Daily XP1350**

**Exercise**

**Exercise**

**One-to-many merge with multiple tables**

In this exercise, assume that you are looking to start a business in the city of Chicago. Your perfect idea is to start a company that uses goats to mow the lawn for other businesses. However, you have to choose a location in the city to put your goat farm. You need a location with a great deal of space and relatively few businesses and people around to avoid complaints about the smell. You will need to merge three tables to help you choose your location. The land\_use table has info on the percentage of vacant land by city ward. The census table has population by ward, and the licenses table lists businesses by ward.

The land\_use, census, and licenses tables have been loaded for you.

**Instructions 1/3**

* Merge land\_use and census on the ward column. Merge the result of this with licenses on the ward column, using the suffix \_cen for the left table and \_lic for the right table. Save this to the variable land\_cen\_lic.
* # Merge land\_use and census and merge result with licenses including suffixes
* land\_cen\_lic = \_\_\_\_

# Merge the ridership, cal, and stations tables

ridership\_cal\_stations = ridership.merge(cal, on=['year','month','day']) \

.merge(stations, on='station\_id')

# Create a filter to filter ridership\_cal\_stations

filter\_criteria = ((ridership\_cal\_stations['month'] == 7)

& (ridership\_cal\_stations['day\_type'] == 'Weekday')

& (ridership\_cal\_stations['station\_name'] == 'Wilson'))

# Use .loc and the filter to select for rides

print(ridership\_cal\_stations.loc[filter\_criteria, 'rides'].sum())

# Merge licenses and zip\_demo, on zip; and merge the wards on ward

licenses\_zip\_ward = licenses.merge(zip\_demo, on= 'zip') \

.merge(wards, on= 'ward')

# Print the results by alderman and show median income

print(licenses\_zip\_ward.groupby('alderman').agg({'income':'median'}))

income

alderman

Ameya Pawar 66246.0

Anthony A. Beale 38206.0

Anthony V. Napolitano 82226.0

Ariel E. Reyboras 41307.0

Brendan Reilly 110215.0

Brian Hopkins 87143.0

Carlos Ramirez-Rosa 66246.0

Carrie M. Austin 38206.0

Chris Taliaferro 55566.0

Daniel "Danny" Solis 41226.0

David H. Moore 33304.0

Deborah Mell 66246.0

Debra L. Silverstein 50554.0

Derrick G. Curtis 65770.0

Edward M. Burke 42335.0

Emma M. Mitts 36283.0

George Cardenas 33959.0

Gilbert Villegas 41307.0

Gregory I. Mitchell 24941.0

Harry Osterman 45442.0

Howard B. Brookins, Jr. 33304.0

James Cappleman 79565.0

Jason C. Ervin 41226.0

Joe Moore 39163.0

John S. Arena 70122.0

Leslie A. Hairston 28024.0

Margaret Laurino 70122.0

Marty Quinn 67045.0

Matthew J. O'Shea 59488.0

Michael R. Zalewski 42335.0

Michael Scott, Jr. 31445.0

Michelle A. Harris 32558.0

Michelle Smith 100116.0

Milagros "Milly" Santiago 41307.0

Nicholas Sposato 62223.0

Pat Dowell 46340.0

Patrick Daley Thompson 41226.0

Patrick J. O'Connor 50554.0

Proco "Joe" Moreno 87143.0

Raymond A. Lopez 33959.0

Ricardo Munoz 31445.0

Roberto Maldonado 68223.0

Roderick T. Sawyer 32558.0

Scott Waguespack 68223.0

Susan Sadlowski Garza 38417.0

Tom Tunney 88708.0

Toni L. Foulkes 27573.0

Walter Burnett, Jr. 87143.0

William D. Burns 107811.0

Willie B. Cochran 28024.0

land\_cen\_lic = land\_use.merge(census, on= 'ward') \

                        .merge(licenses, on= 'ward')

# Merge land\_use and census and merge result with licenses including suffixes

#land\_cen\_lic = land\_use.merge(census, on='ward') \

# .merge(licenses, on= 'ward', suffixes=('\_cen', '\_lic')

land\_cen\_lic = land\_use.merge(census, on= 'ward') \

.merge(licenses, on= 'ward', suffixes=('\_cen', '\_lic'))

* Group land\_cen\_lic by ward, pop\_2010 (the population in 2010), and vacant, then count the number of accounts. Save the results to pop\_vac\_lic.
* # Merge land\_use and census and merge result with licenses including suffixes
* land\_cen\_lic = land\_use.merge(census, on='ward') \
* .merge(licenses, on='ward', suffixes=('\_cen','\_lic'))
* # Group by ward, pop\_2010, and vacant, then count the # of accounts
* pop\_vac\_lic = land\_cen\_lic.groupby(\_\_\_\_,
* as\_index=False).agg({'account':'count'})
* Sort pop\_vac\_lic by vacant, account, andpop\_2010 in descending, ascending, and ascending order respectively. Save it as sorted\_pop\_vac\_lic.
* # Merge land\_use and census and merge result with licenses including suffixes
* land\_cen\_lic = land\_use.merge(census, on='ward') \
* .merge(licenses, on='ward', suffixes=('\_cen','\_lic'))
* # Group by ward, pop\_2010, and vacant, then count the # of accounts
* pop\_vac\_lic = land\_cen\_lic.groupby(['ward','pop\_2010','vacant'],
* as\_index=False).agg({'account':'count'})
* # Sort pop\_vac\_lic and print the results
* sorted\_pop\_vac\_lic = pop\_vac\_lic.sort\_values(\_\_\_\_,
* ascending=\_\_\_\_)
* # Print the top few rows of sorted\_pop\_vac\_lic
* print(sorted\_pop\_vac\_lic.head())

# Merge land\_use and census and merge result with licenses including suffixes

#land\_cen\_lic = land\_use.merge(census, on='ward') \

# .merge(licenses, on= 'ward', suffixes=('\_cen', '\_lic')

land\_cen\_lic = land\_use.merge(census, on= 'ward') \

.merge(licenses, on= 'ward', suffixes=('\_cen', '\_lic'))

# Merge land\_use and census and merge result with licenses including suffixes

land\_cen\_lic = land\_use.merge(census, on='ward') \

.merge(licenses, on='ward', suffixes=('\_cen','\_lic'))

# Group by ward, pop\_2010, and vacant, then count the # of accounts

pop\_vac\_lic = land\_cen\_lic.groupby(['ward', 'pop\_2010', 'vacant'],

as\_index=False).agg({'account':'count'})

ERROR! Session/line number was not unique in database. History logging moved to new session 25

# Merge land\_use and census and merge result with licenses including suffixes

land\_cen\_lic = land\_use.merge(census, on='ward') \

                    .merge(licenses, on='ward', suffixes=('\_cen','\_lic'))

# Group by ward, pop\_2010, and vacant, then count the # of accounts

pop\_vac\_lic = land\_cen\_lic.groupby(['ward','pop\_2010','vacant'],

                                   as\_index=False).agg({'account':'count'})

# Sort pop\_vac\_lic and print the results

sorted\_pop\_vac\_lic = pop\_vac\_lic.sort\_values(['vacant', 'account', 'pop\_2010'],

                                             ascending=[False,True,True])

# Print the top few rows of sorted\_pop\_vac\_lic

print(sorted\_pop\_vac\_lic.head())

# Merge land\_use and census and merge result with licenses including suffixes

land\_cen\_lic = land\_use.merge(census, on='ward') \

.merge(licenses, on='ward', suffixes=('\_cen','\_lic'))

# Group by ward, pop\_2010, and vacant, then count the # of accounts

pop\_vac\_lic = land\_cen\_lic.groupby(['ward','pop\_2010','vacant'],

as\_index=False).agg({'account':'count'})

# Sort pop\_vac\_lic and print the results

sorted\_pop\_vac\_lic = pop\_vac\_lic.sort\_values(['vacant', 'account', 'pop\_2010'],

ascending=[False,True,True])

# Print the top few rows of sorted\_pop\_vac\_lic

print(sorted\_pop\_vac\_lic.head())

ERROR! Session/line number was not unique in database. History logging moved to new session 30

ward pop\_2010 vacant account

47 7 51581 19 80

12 20 52372 15 123

1 10 51535 14 130

16 24 54909 13 98

7 16 51954 13 156

**Daily XP1450**

# Left join

**50 XP**

## 1. Left join

Greetings, and welcome back! In this lesson, we will discuss how a left join works, which is another way to merge two tables. Before we start talking about left joins, let's quickly review what we have learned so far.

## 2. Quick review

In chapter 1, we introduced the pandas merge method that allows us to combine two tables by specifying one or more key columns to link the tables by. By default, the merge method performs an inner join, returning only the rows of data with matching values in the key columns of both tables.

## 3. Left join

In this lesson, we'll talk about the idea of a 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.

## 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. Let's review another example.

## 5. New dataset

To help us learn more about left joins and other concepts in this chapter, we will use data from The Movie Database, a community-built movie database with info on thousands of movies, their casts, and popularity. In our next example, we have two tables from The Movie Database that we want to merge.

## 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.

## 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.

## 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.

## 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.

## 10. Let's practice!

Now, let's practice some.

**Daily XP1500**

**Exercise**

**Exercise**

**Counting missing rows with left join**

The Movie Database is supported by volunteers going out into the world, collecting data, and entering it into the database. This includes financial data, such as movie budget and revenue. If you wanted to know which movies are still missing data, you could use a left join to identify them. Practice using a left join by merging the movies table and the financials table.

The movies and financials tables have been loaded for you.

**Instructions 1/3**

**35 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))

**Question**

What column is likely the best column to merge the two tables on?

**Possible Answers**

* 

on='budget'

* 

on='popularity'

* 

on='id'

# Merge land\_use and census and merge result with licenses including suffixes land\_cen\_lic = land\_use.merge(census, on='ward') \ .merge(licenses, on='ward', suffixes=('\_cen','\_lic')) # Group by ward, pop\_2010, and vacant, then count the # of accounts pop\_vac\_lic = land\_cen\_lic.groupby(['ward','pop\_2010','vacant'], as\_index=False).agg({'account':'count'}) # Sort pop\_vac\_lic and print the results sorted\_pop\_vac\_lic = pop\_vac\_lic.sort\_values(['vacant', 'account', 'pop\_2010'], ascending=[False,True,True]) # Print the top few rows of sorted\_pop\_vac\_lic print(sorted\_pop\_vac\_lic.head())

**Exercise**

**Counting missing rows with left join**

The Movie Database is supported by volunteers going out into the world, collecting data, and entering it into the database. This includes financial data, such as movie budget and revenue. If you wanted to know which movies are still missing data, you could use a left join to identify them. Practice using a left join by merging the movies table and the financials table.

The movies and financials tables have been loaded for you.

**Instructions 2/3**

**35 XP**

* [2](javascript:void(0))
* [3](javascript:void(0))
* Merge the movies table, as the left table, with the financials table using a left join, and save the result to movies\_financials.
* # Merge movies and financials with a left join
* movies\_financials = movies.merge(\_\_\_\_)
* Notes

# Merge land\_use and census and merge result with licenses including suffixes

land\_cen\_lic = land\_use.merge(census, on='ward') \

.merge(licenses, on='ward', suffixes=('\_cen','\_lic'))

# Group by ward, pop\_2010, and vacant, then count the # of accounts

pop\_vac\_lic = land\_cen\_lic.groupby(['ward','pop\_2010','vacant'],

as\_index=False).agg({'account':'count'})

# Sort pop\_vac\_lic and print the results

sorted\_pop\_vac\_lic = pop\_vac\_lic.sort\_values(['vacant', 'account', 'pop\_2010'],

ascending=[False,True,True])

# Print the top few rows of sorted\_pop\_vac\_lic

print(sorted\_pop\_vac\_lic.head())

print(movies)

id title popularity release\_date

0 257 Oliver Twist 20.416 2005-09-23

1 14290 Better Luck Tomorrow 3.877 2002-01-12

2 38365 Grown Ups 38.864 2010-06-24

3 9672 Infamous 3.681 2006-11-16

4 12819 Alpha and Omega 12.301 2010-09-17

... ... ... ... ...

4798 3089 Red River 5.345 1948-08-26

4799 11934 The Hudsucker Proxy 14.189 1994-03-11

4800 13807 Exiled 8.486 2006-09-06

4801 73873 Albert Nobbs 7.802 2011-12-21

4802 11622 Blast from the Past 8.737 1999-02-12

[4803 rows x 4 columns]

print(movies)

print(financials)

id title popularity release\_date

0 257 Oliver Twist 20.416 2005-09-23

1 14290 Better Luck Tomorrow 3.877 2002-01-12

2 38365 Grown Ups 38.864 2010-06-24

3 9672 Infamous 3.681 2006-11-16

4 12819 Alpha and Omega 12.301 2010-09-17

... ... ... ... ...

4798 3089 Red River 5.345 1948-08-26

4799 11934 The Hudsucker Proxy 14.189 1994-03-11

4800 13807 Exiled 8.486 2006-09-06

4801 73873 Albert Nobbs 7.802 2011-12-21

4802 11622 Blast from the Past 8.737 1999-02-12

[4803 rows x 4 columns]

id budget revenue

0 19995 237000000 2.788e+09

1 285 300000000 9.610e+08

2 206647 245000000 8.807e+08

3 49026 250000000 1.085e+09

4 49529 260000000 2.841e+08

... ... ... ...

3224 2292 27000 3.151e+06

3225 692 12000 6.000e+06

3226 36095 20000 9.900e+04

3227 14337 7000 4.248e+05

3228 9367 220000 2.041e+06

[3229 rows x 3 columns]

* Count the number of rows in movies\_financials with a null value in the budget column.

# Merge the movies table with the financials table with a left join

movies\_financials = movies.merge(financials, on='id', how='left')

# Count the number of rows in the budget column that are missing

number\_of\_missing\_fin = movies\_financials['budget']\_\_\_\_

# Print the number of movies missing financials

print(number\_of\_missing\_fin)

# Merge land\_use and census and merge result with licenses including suffixes

land\_cen\_lic = land\_use.merge(census, on='ward') \

.merge(licenses, on='ward', suffixes=('\_cen','\_lic'))

# Group by ward, pop\_2010, and vacant, then count the # of accounts

pop\_vac\_lic = land\_cen\_lic.groupby(['ward','pop\_2010','vacant'],

as\_index=False).agg({'account':'count'})

# Sort pop\_vac\_lic and print the results

sorted\_pop\_vac\_lic = pop\_vac\_lic.sort\_values(['vacant', 'account', 'pop\_2010'],

ascending=[False,True,True])

# Print the top few rows of sorted\_pop\_vac\_lic

print(sorted\_pop\_vac\_lic.head())

print(movies)

id title popularity release\_date

0 257 Oliver Twist 20.416 2005-09-23

1 14290 Better Luck Tomorrow 3.877 2002-01-12

2 38365 Grown Ups 38.864 2010-06-24

3 9672 Infamous 3.681 2006-11-16

4 12819 Alpha and Omega 12.301 2010-09-17

... ... ... ... ...

4798 3089 Red River 5.345 1948-08-26

4799 11934 The Hudsucker Proxy 14.189 1994-03-11

4800 13807 Exiled 8.486 2006-09-06

4801 73873 Albert Nobbs 7.802 2011-12-21

4802 11622 Blast from the Past 8.737 1999-02-12

[4803 rows x 4 columns]

print(movies)

print(financials)

id title popularity release\_date

0 257 Oliver Twist 20.416 2005-09-23

1 14290 Better Luck Tomorrow 3.877 2002-01-12

2 38365 Grown Ups 38.864 2010-06-24

3 9672 Infamous 3.681 2006-11-16

4 12819 Alpha and Omega 12.301 2010-09-17

... ... ... ... ...

4798 3089 Red River 5.345 1948-08-26

4799 11934 The Hudsucker Proxy 14.189 1994-03-11

4800 13807 Exiled 8.486 2006-09-06

4801 73873 Albert Nobbs 7.802 2011-12-21

4802 11622 Blast from the Past 8.737 1999-02-12

[4803 rows x 4 columns]

id budget revenue

0 19995 237000000 2.788e+09

1 285 300000000 9.610e+08

2 206647 245000000 8.807e+08

3 49026 250000000 1.085e+09

4 49529 260000000 2.841e+08

... ... ... ...

3224 2292 27000 3.151e+06

3225 692 12000 6.000e+06

3226 36095 20000 9.900e+04

3227 14337 7000 4.248e+05

3228 9367 220000 2.041e+06

[3229 rows x 3 columns]

# Merge movies and financials with a left join

movies\_financials = movies.merge(financials, on='id', how='left')

print(movies\_financials)

id title popularity release\_date budget revenue

0 257 Oliver Twist 20.416 2005-09-23 5.000e+07 4.209e+07

1 14290 Better Luck Tomorrow 3.877 2002-01-12 NaN NaN

2 38365 Grown Ups 38.864 2010-06-24 8.000e+07 2.714e+08

3 9672 Infamous 3.681 2006-11-16 1.300e+07 1.151e+06

4 12819 Alpha and Omega 12.301 2010-09-17 2.000e+07 3.930e+07

... ... ... ... ... ... ...

4798 3089 Red River 5.345 1948-08-26 3.000e+06 9.012e+06

4799 11934 The Hudsucker Proxy 14.189 1994-03-11 NaN NaN

4800 13807 Exiled 8.486 2006-09-06 NaN NaN

4801 73873 Albert Nobbs 7.802 2011-12-21 8.000e+06 5.635e+06

4802 11622 Blast from the Past 8.737 1999-02-12 3.500e+07 4.026e+07

[4803 rows x 6 columns]

<script.py> output:

id title popularity release\_date budget revenue

0 257 Oliver Twist 20.416 2005-09-23 5.000e+07 4.209e+07

1 14290 Better Luck Tomorrow 3.877 2002-01-12 NaN NaN

2 38365 Grown Ups 38.864 2010-06-24 8.000e+07 2.714e+08

3 9672 Infamous 3.681 2006-11-16 1.300e+07 1.151e+06

4 12819 Alpha and Omega 12.301 2010-09-17 2.000e+07 3.930e+07

... ... ... ... ... ... ...

4798 3089 Red River 5.345 1948-08-26 3.000e+06 9.012e+06

4799 11934 The Hudsucker Proxy 14.189 1994-03-11 NaN NaN

4800 13807 Exiled 8.486 2006-09-06 NaN NaN

4801 73873 Albert Nobbs 7.802 2011-12-21 8.000e+06 5.635e+06

4802 11622 Blast from the Past 8.737 1999-02-12 3.500e+07 4.026e+07

[4803 rows x 6 columns]

# Merge the movies table with the financials table with a left join

movies\_financials = movies.merge(financials, on='id', how='left')

# Count the number of rows in the budget column that are missing

number\_of\_missing\_fin = movies\_financials['budget'].isnull().sum()

# Print the number of movies missing financials

print(number\_of\_missing\_fin)

# Merge the movies table with the financials table with a left join

movies\_financials = movies.merge(financials, on='id', how='left')

# Count the number of rows in the budget column that are missing

number\_of\_missing\_fin = movies\_financials['budget'].value\_counts()

# Print the number of movies missing financials

print(number\_of\_missing\_fin)

2.000e+07 126

2.500e+07 115

3.000e+07 114

4.000e+07 111

1.500e+07 107

...

1.240e+07 1

3.500e+06 1

2.115e+07 1

4.168e+07 1

3.730e+06 1

Name: budget, Length: 385, dtype: int64

<script.py> output:

2.000e+07 126

2.500e+07 115

3.000e+07 114

4.000e+07 111

1.500e+07 107

...

1.240e+07 1

3.500e+06 1

2.115e+07 1

4.168e+07 1

3.730e+06 1

Name: budget, Length: 385, dtype: int64

# Merge the movies table with the financials table with a left join

movies\_financials = movies.merge(financials, on='id', how='left')

# Count the number of rows in the budget column that are missing

number\_of\_missing\_fin = movies\_financials['budget'].isnull()

# Print the number of movies missing financials

print(number\_of\_missing\_fin)

0 False

1 True

2 False

3 False

4 False

...

4798 False

4799 True

4800 True

4801 False

4802 False

Name: budget, Length: 4803, dtype: bool

<script.py> output:

0 False

1 True

2 False

3 False

4 False

...

4798 False

4799 True

4800 True

4801 False

4802 False

Name: budget, Length: 4803, dtype: bool

# Merge the movies table with the financials table with a left join

movies\_financials = movies.merge(financials, on='id', how='left')

# Count the number of rows in the budget column that are missing

number\_of\_missing\_fin = movies\_financials['budget'].isnull().sum()

# Print the number of movies missing financials

print(number\_of\_missing\_fin)

1574

<script.py> output:

1574

**Daily XP1600**

**Exercise**

**Exercise**

**Enriching a dataset**

Setting how='left' with the .merge()method is a useful technique for enriching or enhancing a dataset with additional information from a different table. In this exercise, you will start off with a sample of movie data from the movie series *Toy Story*. Your goal is to enrich this data by adding the marketing tag line for each movie. You will compare the results of a left join versus an inner join.

The toy\_story DataFrame contains the *Toy Story* movies. The toy\_story and taglines DataFrames have been loaded for you.

**Instructions 1/2**

**50 XP**

* [1](javascript:void(0))
  + Merge toy\_story and taglines on the id column with a **left join**, and save the result as toystory\_tag.

 [2](javascript:void(0))

* With toy\_story as the left table, merge to it taglines on the id column with an **inner join**, and save as toystory\_tag.
* # Merge the toy\_story and taglines tables with a left join
* toystory\_tag = toy\_story.merge(\_\_\_\_)
* # Print the rows and shape of toystory\_tag
* print(toystory\_tag)
* print(toystory\_tag.shape)

# Merge the movies table with the financials table with a left join movies\_financials = movies.merge(financials, on='id', how='left') # Count the number of rows in the budget column that are missing number\_of\_missing\_fin = movies\_financials['budget'].isnull().sum() # Print the number of movies missing financials print(number\_of\_missing\_fin)

# Merge the toy\_story and taglines tables with a left join

toystory\_tag = toy\_story.merge(taglines, on='id', how='left')

# Print the rows and shape of toystory\_tag

print(toystory\_tag)

print(toystory\_tag.shape)

# Merge the toy\_story and taglines tables with a left join

toystory\_tag = toy\_story.merge(taglines, on='id', how='left')

# Print the rows and shape of toystory\_tag

print(toystory\_tag)

print(toystory\_tag.shape)

id title popularity release\_date tagline

0 10193 Toy Story 3 59.995 2010-06-16 No toy gets left behind.

1 863 Toy Story 2 73.575 1999-10-30 The toys are back!

2 862 Toy Story 73.640 1995-10-30 NaN

(3, 5)

# Merge the toy\_story and taglines tables with a inner join

toystory\_tag = toy\_story.merge(taglines, on='id')

# Print the rows and shape of toystory\_tag

print(toystory\_tag)

print(toystory\_tag.shape)

# Merge the toy\_story and taglines tables with a inner join

toystory\_tag = toy\_story.merge(taglines, on='id')

# Print the rows and shape of toystory\_tag

print(toystory\_tag)

print(toystory\_tag.shape)

id title popularity release\_date tagline

0 10193 Toy Story 3 59.995 2010-06-16 No toy gets left behind.

1 863 Toy Story 2 73.575 1999-10-30 The toys are back!

(2, 5)

**Daily XP1700**

**Exercise**

**Exercise**

**How many rows with a left join?**

Select the **true** statement about left joins.

Try running the following code statements in the IPython shell.

* left\_table.merge(one\_to\_one, on='id', how='left').shape
* left\_table.merge(one\_to\_many, on='id', how='left').shape

Note that the left\_table starts out with **4** rows.

**Instructions**

**50 XP**

**Possible Answers**

* 

The output of a **one-to-one** merge with a left join will have **more** rows than the left table.

* 

The output of a **one-to-one** merge with a left join will have **fewer** rows than the left table.

* 

The output of a **one-to-many** merge with a left join will have greater than or equal rows than the left table.

# Merge the toy\_story and taglines tables with a inner join

toystory\_tag = toy\_story.merge(taglines, on='id')

# Print the rows and shape of toystory\_tag

print(toystory\_tag)

print(toystory\_tag.shape)

In [1]:

print(left\_table.merge(one\_to\_one, on='id', how='left').shape)

(4, 5)

In [2]:

print(left\_table.merge(one\_to\_many, on='id', how='left').shape)

(232, 6)

**Daily XP1750**

# Other joins

**50 XP**

## 1. Other joins

All right, let's continue on. You now know how to use the merge method to perform an inner and left join. The merge method supports two other join types.

## 2. Right join

Let's start with the right join. It 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.

## 3. Right join

These example tables show the result of a right join. Only rows from the left table where the column C matches are returned. Where there isn't a match, the columns from the left table will be missing in the result table, like rows one and four.

## 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.

## 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.

## 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.

## 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. We could explore further. However, let's move on to our last type of 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.

## 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.

## 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.

## 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.

## 12. Let's practice!

Let's practice!

**Daily XP50**

**Exercise**

**Exercise**

**Right join to find unique movies**

Most of the recent big-budget science fiction movies can also be classified as action movies. You are given a table of science fiction movies called scifi\_movies and another table of action movies called action\_movies. Your goal is to find which movies are considered only science fiction movies. Once you have this table, you can merge the movies table in to see the movie names. Since this exercise is related to science fiction movies, use a right join as your superhero power to solve this problem.

The movies, scifi\_movies, and action\_movies tables have been loaded for you.

**Instructions 1/4**

**25 XP**

* [1](javascript:void(0))
  + Merge action\_movies and scifi\_movies tables with a **right join** on movie\_id. Save the result as action\_scifi.

 [2](javascript:void(0))

* Update the merge to add suffixes, where '\_act' and '\_sci' are suffixes for the left and right tables, respectively.

 [3](javascript:void(0))

* From action\_scifi, subset only the rows where the genre\_act column is null.

 [4](javascript:void(0))

* Merge movies and scifi\_only using the id column in the left table and the movie\_id column in the right table with an inner join.
* # Merge action\_movies to scifi\_movies with right join
* action\_scifi = \_\_\_\_.merge(\_\_\_\_)

# Merge the toy\_story and taglines tables with a inner join toystory\_tag = toy\_story.merge(taglines, on='id') # Print the rows and shape of toystory\_tag print(toystory\_tag) print(toystory\_tag.shape)

# Merge action\_movies to scifi\_movies with right join

action\_scifi = action\_movies.merge(scifi\_movies, on='movie\_id', how='right',

                                   \_\_\_\_)

# Print the first few rows of action\_scifi to see the structure

print(action\_scifi.head())

# Merge action\_movies to scifi\_movies with right join

action\_scifi = action\_movies.merge(scifi\_movies, on='movie\_id', how='right')

print(action\_scifi)

movie\_id genre\_x genre\_y

0 11 Action Science Fiction

1 18 Action Science Fiction

2 19 NaN Science Fiction

3 38 NaN Science Fiction

4 62 NaN Science Fiction

.. ... ... ...

530 335866 NaN Science Fiction

531 347548 NaN Science Fiction

532 360188 NaN Science Fiction

533 367551 Action Science Fiction

534 371690 NaN Science Fiction

[535 rows x 3 columns]

<script.py> output:

movie\_id genre\_x genre\_y

0 11 Action Science Fiction

1 18 Action Science Fiction

2 19 NaN Science Fiction

3 38 NaN Science Fiction

4 62 NaN Science Fiction

.. ... ... ...

530 335866 NaN Science Fiction

531 347548 NaN Science Fiction

532 360188 NaN Science Fiction

533 367551 Action Science Fiction

534 371690 NaN Science Fiction

[535 rows x 3 columns]

# Merge action\_movies to the scifi\_movies with right join

action\_scifi = action\_movies.merge(scifi\_movies, on='movie\_id', how='right',

                                   suffixes=('\_act','\_sci'))

# From action\_scifi, select only the rows where the genre\_act column is null

scifi\_only = action\_scifi[\_\_\_\_]

# Merge action\_movies to scifi\_movies with right join

action\_scifi = action\_movies.merge(scifi\_movies, on='movie\_id', how='right',

suffixes=('\_act', '\_sci'))

# Print the first few rows of action\_scifi to see the structure

print(action\_scifi.head())

movie\_id genre\_act genre\_sci

0 11 Action Science Fiction

1 18 Action Science Fiction

2 19 NaN Science Fiction

3 38 NaN Science Fiction

4 62 NaN Science Fiction

<script.py> output:

movie\_id genre\_act genre\_sci

0 11 Action Science Fiction

1 18 Action Science Fiction

2 19 NaN Science Fiction

3 38 NaN Science Fiction

4 62 NaN Science Fiction

# Merge action\_movies to the scifi\_movies with right join

action\_scifi = action\_movies.merge(scifi\_movies, on='movie\_id', how='right',

suffixes=('\_act','\_sci'))

# From action\_scifi, select only the rows where the genre\_act column is null

scifi\_only = action\_scifi[action\_scifi['genre\_act'].isna()]

print(scifi\_only)

movie\_id genre\_act genre\_sci

2 19 NaN Science Fiction

3 38 NaN Science Fiction

4 62 NaN Science Fiction

5 68 NaN Science Fiction

6 74 NaN Science Fiction

.. ... ... ...

529 333371 NaN Science Fiction

530 335866 NaN Science Fiction

531 347548 NaN Science Fiction

532 360188 NaN Science Fiction

534 371690 NaN Science Fiction

[258 rows x 3 columns]

<script.py> output:

movie\_id genre\_act genre\_sci

2 19 NaN Science Fiction

3 38 NaN Science Fiction

4 62 NaN Science Fiction

5 68 NaN Science Fiction

6 74 NaN Science Fiction

.. ... ... ...

529 333371 NaN Science Fiction

530 335866 NaN Science Fiction

531 347548 NaN Science Fiction

532 360188 NaN Science Fiction

534 371690 NaN Science Fiction

[258 rows x 3 columns]

# Merge action\_movies to the scifi\_movies with right join

action\_scifi = action\_movies.merge(scifi\_movies, on='movie\_id', how='right',

                                   suffixes=('\_act','\_sci'))

# From action\_scifi, select only the rows where the genre\_act column is null

scifi\_only = action\_scifi[action\_scifi['genre\_act'].isnull()]

# Merge the movies and scifi\_only tables with an inner join

movies\_and\_scifi\_only = movies.\_\_\_\_

# Print the first few rows and shape of movies\_and\_scifi\_only

print(movies\_and\_scifi\_only.head())

print(movies\_and\_scifi\_only.shape)

# Merge action\_movies to the scifi\_movies with right join

action\_scifi = action\_movies.merge(scifi\_movies, on='movie\_id', how='right',

                                   suffixes=('\_act','\_sci'))

# From action\_scifi, select only the rows where the genre\_act column is null

scifi\_only = action\_scifi[action\_scifi['genre\_act'].isnull()]

# Merge the movies and scifi\_only tables with an inner join

movies\_and\_scifi\_only = movies.merge(scifi\_only, left\_on='id', right\_on='movie\_id')

# Print the first few rows and shape of movies\_and\_scifi\_only

print(movies\_and\_scifi\_only.head())

print(movies\_and\_scifi\_only.shape)

# Merge action\_movies to the scifi\_movies with right join

action\_scifi = action\_movies.merge(scifi\_movies, on='movie\_id', how='right',

suffixes=('\_act','\_sci'))

# From action\_scifi, select only the rows where the genre\_act column is null

scifi\_only = action\_scifi[action\_scifi['genre\_act'].isnull()]

# Merge the movies and scifi\_only tables with an inner join

movies\_and\_scifi\_only = movies.merge(scifi\_only, left\_on='id', right\_on='movie\_id')

# Print the first few rows and shape of movies\_and\_scifi\_only

print(movies\_and\_scifi\_only.head())

print(movies\_and\_scifi\_only.shape)

id title popularity release\_date movie\_id genre\_act genre\_sci

0 18841 The Lost Skeleton of Cadavra 1.681 2001-09-12 18841 NaN Science Fiction

1 26672 The Thief and the Cobbler 2.439 1993-09-23 26672 NaN Science Fiction

2 15301 Twilight Zone: The Movie 12.903 1983-06-24 15301 NaN Science Fiction

3 8452 The 6th Day 18.447 2000-11-17 8452 NaN Science Fiction

4 1649 Bill & Ted's Bogus Journey 11.350 1991-07-19 1649 NaN Science Fiction

(258, 7)

<script.py> output:

id title popularity release\_date movie\_id genre\_act genre\_sci

0 18841 The Lost Skeleton of Cadavra 1.681 2001-09-12 18841 NaN Science Fiction

1 26672 The Thief and the Cobbler 2.439 1993-09-23 26672 NaN Science Fiction

2 15301 Twilight Zone: The Movie 12.903 1983-06-24 15301 NaN Science Fiction

3 8452 The 6th Day 18.447 2000-11-17 8452 NaN Science Fiction

4 1649 Bill & Ted's Bogus Journey 11.350 1991-07-19 1649 NaN Science Fiction

(258, 7)

**Daily XP150**

**Exercise**

**Exercise**

**Popular genres with right join**

What are the genres of the most popular movies? To answer this question, you need to merge data from the movies and movie\_to\_genres tables. In a table called pop\_movies, the top 10 most popular movies in the movies table have been selected. To ensure that you are analyzing all of the popular movies, merge it with the movie\_to\_genres table using a right join. To complete your analysis, count the number of different genres. Also, the two tables can be merged by the movie ID. However, in pop\_movies that column is called id, and in movies\_to\_genres it's called movie\_id.

The pop\_movies and movie\_to\_genres tables have been loaded for you.

**Instructions**

**100 XP**

* Merge movie\_to\_genres and pop\_movies using a right join. Save the results as genres\_movies.
* Group genres\_movies by genre and count the number of id values.
* # Use right join to merge the movie\_to\_genres and pop\_movies tables
* genres\_movies = \_\_\_\_.merge(\_\_\_\_, how='\_\_\_\_',
* \_\_\_\_,
* \_\_\_\_)
* # Count the number of genres
* genre\_count = genres\_movies.groupby('\_\_\_\_').agg({'id':'count'})
* # Plot a bar chart of the genre\_count
* genre\_count.plot(kind='bar')
* plt.show()

# Merge action\_movies to the scifi\_movies with right join action\_scifi = action\_movies.merge(scifi\_movies, on='movie\_id', how='right', suffixes=('\_act','\_sci')) # From action\_scifi, select only the rows where the genre\_act column is null scifi\_only = action\_scifi[action\_scifi['genre\_act'].isnull()] # Merge the movies and scifi\_only tables with an inner join movies\_and\_scifi\_only = movies.merge(scifi\_only, left\_on='id', right\_on='movie\_id') # Print the first few rows and shape of movies\_and\_scifi\_only print(movies\_and\_scifi\_only.head()) print(movies\_and\_scifi\_only.shape)

# Use right join to merge the movie\_to\_genres and pop\_movies tables

genres\_movies = movie\_to\_genres.merge(pop\_movies, how='right',

                                      left\_on='movie\_id',

                                      right\_on='id')

# Count the number of genres

genre\_count = genres\_movies.groupby('genre').agg({'id':'count'})

# Plot a bar chart of the genre\_count

genre\_count.plot(kind='bar')

plt.show()

# Use right join to merge the movie\_to\_genres and pop\_movies tables genres\_movies = movie\_to\_genres.merge(pop\_movies, how='right', left\_on='movie\_id', right\_on='id') # Count the number of genres genre\_count = genres\_movies.groupby('genre').agg({'id':'count'}) # Plot a bar chart of the genre\_count genre\_count.plot(kind='bar') plt.show()

**Daily XP250**

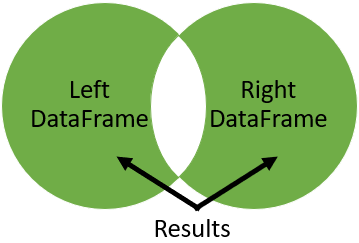
**Exercise**

**Exercise**

**Using outer join to select actors**

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. To try for yourself, you have been given two tables with a list of actors from two popular movies: *Iron Man 1* and *Iron Man 2*. Most of the actors played in both movies. Use an outer join to find actors who ***did not*** act in both movies.

The *Iron Man 1* table is called iron\_1\_actors, and *Iron Man 2* table is called iron\_2\_actors. Both tables have been loaded for you and a few rows printed so you can see the structure.



**Instructions**

**100 XP**

* Save to iron\_1\_and\_2 the merge of iron\_1\_actors (left) with iron\_2\_actors tables with an outer join on the id column, and set suffixes to ('\_1','\_2').
* Create an index that returns True if name\_1 or name\_2 are null, and False otherwise.

# Merge iron\_1\_actors to iron\_2\_actors on id with outer join using suffixes

iron\_1\_and\_2 = iron\_1\_actors.merge(\_\_\_\_,

                                     \_\_\_\_,

                                     \_\_\_\_,

                                     suffixes=\_\_\_\_)

# Create an index that returns true if name\_1 or name\_2 are null

m = ((iron\_1\_and\_2['name\_1'].\_\_\_\_) |

     (iron\_1\_and\_2['\_\_\_\_'].\_\_\_\_))

# Print the first few rows of iron\_1\_and\_2

print(iron\_1\_and\_2[m].head())

iron\_1\_actors #########

character id name

3 Yinsen 17857 Shaun Toub

4 Virginia "Pepper" Potts 12052 Gwyneth Paltrow

iron\_2\_actors #########

character id name

4 Ivan Vanko / Whiplash 2295 Mickey Rourke

3 Natalie Rushman / Natasha Romanoff / Black Widow 1245 Scarlett Johansson

# Merge iron\_1\_actors to iron\_2\_actors on id with outer join using suffixes

iron\_1\_and\_2 = iron\_1\_actors.merge(iron\_2\_actors,

                                     on='id',

                                     how='outer',

                                     suffixes=('\_1','\_2'))

# Create an index that returns true if name\_1 or name\_2 are null

m = ((iron\_1\_and\_2['name\_1'].isnull()) |

     (iron\_1\_and\_2['name\_2'].isnull()))

# Print the first few rows of iron\_1\_and\_2

print(iron\_1\_and\_2[m].head())

iron\_1\_actors #########

character id name

3 Yinsen 17857 Shaun Toub

4 Virginia "Pepper" Potts 12052 Gwyneth Paltrow

iron\_2\_actors #########

character id name

4 Ivan Vanko / Whiplash 2295 Mickey Rourke

3 Natalie Rushman / Natasha Romanoff / Black Widow 1245 Scarlett Johansson

# Merge iron\_1\_actors to iron\_2\_actors on id with outer join using suffixes

iron\_1\_and\_2 = iron\_1\_actors.merge(iron\_2\_actors,

on='id',

how='outer',

suffixes=('\_1','\_2'))

# Create an index that returns true if name\_1 or name\_2 are null

m = ((iron\_1\_and\_2['name\_1'].isnull()) |

(iron\_1\_and\_2['name\_2'].isnull()))

# Print the first few rows of iron\_1\_and\_2

print(iron\_1\_and\_2[m].head())**Daily XP350**

# Merging a table to itself

**50 XP**

## 1. Merging a table to itself

Hello again! In this lesson, we will talk about merging a table to itself. This type of merge is also referred to as a self join. So, let's get started.

## 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.

## 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.

## 4. Merging a table to itself

To complete this merge, we set the sequels table as input to the merge method for both the left and right tables. 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.

## 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.

## 6. Merging a table to itself with left join

Pausing here is a good time to highlight again that when merging a table to itself, 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.

## 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.

## 8. Let's practice!

Alright, let's practice merging a table to itself.

**Daily XP400**

**Exercise**

**Exercise**

**Self join**

Merging a table to itself can be useful when you want to compare values in a column to other values in the same column. In this exercise, you will practice this by creating a table that for each movie will list the movie director and a member of the crew on one row. You have been given a table called crews, which has columns id, job, and name. First, merge the table to itself using the movie ID. This merge will give you a larger table where for each movie, every job is matched against each other. Then select only those rows with a director in the left table, and avoid having a row where the director's job is listed in both the left and right tables. This filtering will remove job combinations that aren't with the director.

The crews table has been loaded for you.

**Instructions 1/3**

**35 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))
* To a variable called crews\_self\_merged, merge the crews table to itself on the id column using an inner join, setting the suffixes to '\_dir' and '\_crew' for the left and right tables respectively.
* # Merge the crews table to itself
* crews\_self\_merged = \_\_\_\_

crews\_self\_merged = crews.merge(crews, on='id', suffixes=('\_dir', '\_crew'))

print(crews\_self\_merged)

Merge the crews table to itself

crews\_self\_merged = crews.merge(crews, on='id', suffixes=('\_dir', '\_crew'))

print(crews\_self\_merged)

id job\_dir name\_dir job\_crew name\_crew

0 19995 Editor Stephen E. Rivkin Editor Stephen E. Rivkin

1 19995 Editor Stephen E. Rivkin Sound Designer Christopher Boyes

2 19995 Editor Stephen E. Rivkin Casting Mali Finn

3 19995 Editor Stephen E. Rivkin Director James Cameron

4 19995 Editor Stephen E. Rivkin Writer James Cameron

... ... ... ... ... ...

834189 25975 Director Jon Gunn Director Brett Winn

834190 25975 Director Brett Winn Executive Producer Clark Peterson

834191 25975 Director Brett Winn Director Brian Herzlinger

834192 25975 Director Brett Winn Director Jon Gunn

834193 25975 Director Brett Winn Director Brett Winn

[834194 rows x 5 columns]

* Create a Boolean index, named boolean\_filter, that selects rows from the left table with the *job* of 'Director' and avoids rows with the *job* of 'Director' in the right table.

# Merge the crews table to itself

crews\_self\_merged = crews.merge(crews, on='id', how='inner',

                                suffixes=('\_dir','\_crew'))

# Create a Boolean index to select the appropriate

boolean\_filter = ((crews\_self\_merged['job\_dir'] == \_\_\_\_) &

     (crews\_self\_merged['\_\_\_\_'] != \_\_\_\_))

direct\_crews = crews\_self\_merged[boolean\_filter]

# Merge the crews table to itself

crews\_self\_merged = crews.merge(crews, on='id', how='inner',

                                suffixes=('\_dir','\_crew'))

# Create a Boolean index to select the appropriate

boolean\_filter = ((crews\_self\_merged['job\_dir'] == 'Director') &

     (crews\_self\_merged['job\_crew'] != 'Director'))

direct\_crews = crews\_self\_merged[boolean\_filter]

print(direct\_crews)

# Merge the crews table to itself

crews\_self\_merged = crews.merge(crews, on='id', how='inner',

suffixes=('\_dir','\_crew'))

# Create a Boolean index to select the appropriate

boolean\_filter = ((crews\_self\_merged['job\_dir'] == 'Director') &

(crews\_self\_merged['job\_crew'] != 'Director'))

direct\_crews = crews\_self\_merged[boolean\_filter]

print(direct\_crews)

id job\_dir name\_dir job\_crew name\_crew

156 19995 Director James Cameron Editor Stephen E. Rivkin

157 19995 Director James Cameron Sound Designer Christopher Boyes

158 19995 Director James Cameron Casting Mali Finn

160 19995 Director James Cameron Writer James Cameron

161 19995 Director James Cameron Set Designer Richard F. Mays

... ... ... ... ... ...

834166 72766 Director Edward Burns Editor Janet Gaynor

834174 231617 Director Scott Smith Executive Producer Scott Smith

834182 25975 Director Brian Herzlinger Executive Producer Clark Peterson

834186 25975 Director Jon Gunn Executive Producer Clark Peterson

834190 25975 Director Brett Winn Executive Producer Clark Peterson

[40845 rows x 5 columns]

Use the .head() method to print the first few rows of direct\_crews.

# Merge the crews table to itself

crews\_self\_merged = crews.merge(crews, on='id', how='inner',

                                suffixes=('\_dir','\_crew'))

# Create a boolean index to select the appropriate rows

boolean\_filter = ((crews\_self\_merged['job\_dir'] == 'Director') &

                  (crews\_self\_merged['job\_crew'] != 'Director'))

direct\_crews = crews\_self\_merged[boolean\_filter]

# Print the first few rows of direct\_crews

print(\_\_\_\_)

Great job! By merging the table to itself, you compared the value of the \_\_director\_\_ from the jobs column to other values from the jobs column. With the output, you can quickly see different movie directors and the people they worked with in the same movie.

**How does pandas handle self joins?**

Select the **false** statement about merging a table to itself.

**Answer the question**

**50XP**

**Possible Answers**

* 

You can merge a table to itself with a right join.

press1

* 

Merging a table to itself can allow you to compare values in a column to other values in the same column.

press2

* 

The pandas module limits you to one merge where you merge a table to itself. You cannot repeat this process over and over.

**This is False statement and the answer**

press3

* 

Merging a table to itself is like working with two separate tables.

press4

Perfect! This statement is **false**. pandas treats a merge of a table to itself the same as any other merge. Therefore, it does not limit you from chaining multiple .merge() methods together.

character\_1 id name\_1 character\_2 name\_2

0 Yinsen 17857 Shaun Toub NaN NaN

2 Obadiah Stane / Iron Monger 1229 Jeff Bridges NaN NaN

3 War Machine 18288 Terrence Howard NaN NaN

5 Raza 57452 Faran Tahir NaN NaN

8 Abu Bakaar 173810 Sayed Badreya NaN NaN

**Daily XP550**

# Merging on indexes

**50 XP**

## 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.

## 2. Table with an index

Here, we show the movies table that was introduced earlier in this chapter. The index is the default 0, 1, 2, 3, etc., auto-increment. In this second version, the id column is the index for the table.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 10. Let's practice!

Now it's time for you try out a few exercises.

**Daily XP50**

**Exercise**

**Exercise**

**Index merge for movie ratings**

To practice merging on indexes, you will merge movies and a table called ratings that holds info about movie ratings. Make sure your merge returns **all** of the rows from the movies table and not all the rows of ratings table need to be included in the result.

The movies and ratings tables have been loaded for you.

**Instructions**

**100 XP**

* Merge movies and ratings on the index and save to a variable called movies\_ratings, ensuring that all of the rows from the movies table are returned.

# Merge to the movies table the ratings table on the index

movies\_ratings = \_\_\_\_

# Print the first few rows of movies\_ratings

print(movies\_ratings.head())

# Merge the crews table to itself crews\_self\_merged = crews.merge(crews, on='id', how='inner', suffixes=('\_dir','\_crew')) # Create a boolean index to select the appropriate rows boolean\_filter = ((crews\_self\_merged['job\_dir'] == 'Director') & (crews\_self\_merged['job\_crew'] != 'Director')) direct\_crews = crews\_self\_merged[boolean\_filter] # Print the first few rows of direct\_crews print(direct\_crews.head())

# Merge to the movies table the ratings table on the index

movies\_ratings = movies.merge(ratings, on='id')

print(ratings)

# Print the first few rows of movies\_ratings

print(movies\_ratings.head())

# Merge to the movies table the ratings table on the index

movies\_ratings = movies.merge(ratings, on='id')

# Print the first few rows of movies\_ratings

print(movies\_ratings.head())

title popularity release\_date vote\_average vote\_count

id

257 Oliver Twist 20.416 2005-09-23 6.7 274.0

14290 Better Luck Tomorrow 3.877 2002-01-12 6.5 27.0

38365 Grown Ups 38.864 2010-06-24 6.0 1705.0

9672 Infamous 3.681 2006-11-16 6.4 60.0

12819 Alpha and Omega 12.301 2010-09-17 5.3 124.0

# Merge to the movies table the ratings table on the index

movies\_ratings = movies.merge(ratings, on='id')

print(ratings)

# Print the first few rows of movies\_ratings

print(movies\_ratings.head())

vote\_average vote\_count

id

19995 7.2 11800.0

285 6.9 4500.0

206647 6.3 4466.0

49026 7.6 9106.0

49529 6.1 2124.0

... ... ...

9367 6.6 238.0

72766 5.9 5.0

231617 7.0 6.0

126186 5.7 7.0

25975 6.3 16.0

[4803 rows x 2 columns]

title popularity release\_date vote\_average vote\_count

id

257 Oliver Twist 20.416 2005-09-23 6.7 274.0

14290 Better Luck Tomorrow 3.877 2002-01-12 6.5 27.0

38365 Grown Ups 38.864 2010-06-24 6.0 1705.0

9672 Infamous 3.681 2006-11-16 6.4 60.0

12819 Alpha and Omega 12.301 2010-09-17 5.3 124.0

<script.py> output:

vote\_average vote\_count

id

19995 7.2 11800.0

285 6.9 4500.0

206647 6.3 4466.0

49026 7.6 9106.0

49529 6.1 2124.0

... ... ...

9367 6.6 238.0

72766 5.9 5.0

231617 7.0 6.0

126186 5.7 7.0

25975 6.3 16.0

[4803 rows x 2 columns]

title popularity release\_date vote\_average vote\_count

id

257 Oliver Twist 20.416 2005-09-23 6.7 274.0

14290 Better Luck Tomorrow 3.877 2002-01-12 6.5 27.0

38365 Grown Ups 38.864 2010-06-24 6.0 1705.0

9672 Infamous 3.681 2006-11-16 6.4 60.0

12819 Alpha and Omega 12.301 2010-09-17 5.3 124.0

Good work! 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.

**Daily XP150**

**Exercise**

**Exercise**

**Do sequels earn more?**

It is time to put together many of the aspects that you have learned in this chapter. In this exercise, you'll find out which movie sequels earned the most compared to the original movie. To answer this question, you will merge a modified version of the sequels and financials tables where their index is the movie ID. You will need to choose a merge type that will return all of the rows from the sequels table and not all the rows of financials table need to be included in the result. From there, you will join the resulting table to itself so that you can compare the revenue values of the original movie to the sequel. Next, you will calculate the difference between the two revenues and sort the resulting dataset.

The sequels and financials tables have been provided.

**Instructions 1/4**

**25 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))
* [4](javascript:void(0))
* With the sequels table on the left, merge to it the financials table on index named id, ensuring that all the rows from the sequels are returned and some rows from the other table may not be returned, Save the results to sequels\_fin.
* # Merge sequels and financials on index id
* sequels\_fin = \_\_\_\_

# Merge sequels and financials on index id #print(sequels) #print(financials) sequels\_fin = sequels.merge(financials, on='id', how='left') #print(sequels\_fin)

Merge the sequels\_fin table to itself with an inner join, where the left and right tables merge on sequel and id respectively wit# Merge sequels and financials on index id

sequels\_fin = sequels.merge(financials, on='id', how='left')

# Self merge with suffixes as inner join with left on sequel and right on id

orig\_seq = \_\_\_\_.merge(\_\_\_\_, how=\_\_\_\_, left\_on=\_\_\_\_,

                             right\_on=\_\_\_\_, right\_index=\_\_\_\_,

                             suffixes=\_\_\_\_)

# Add calculation to subtract revenue\_org from revenue\_seq

orig\_seq['diff'] = orig\_seq['revenue\_seq'] - orig\_seq['revenue\_org']

Merge sequels and financials on index id

#print(sequels)

#print(financials)

sequels\_fin = sequels.merge(financials, on='id', how='left')

#print(sequels\_fin)

# Merge sequels and financials on index id

sequels\_fin = sequels.merge(financials, on='id', how='left')

# Self merge with suffixes as inner join with left on sequel and right on id

orig\_seq = sequels\_fin.merge(sequels\_fin, how='inner', left\_on='sequel',

right\_on='id', right\_index=True,

suffixes=('\_org', '\_seq'))

# Add calculation to subtract revenue\_org from revenue\_seq

orig\_seq['diff'] = orig\_seq['revenue\_seq'] - orig\_seq['revenue\_org']

Select the title\_org, title\_seq, and diff columns of orig\_seq and save this as titles\_diff.

# Merge sequels and financials on index id

sequels\_fin = sequels.merge(financials, on='id', how='left')

# Self merge with suffixes as inner join with left on sequel and right on id

orig\_seq = sequels\_fin.merge(sequels\_fin, how='inner', left\_on='sequel',

                             right\_on='id', right\_index=True,

                             suffixes=('\_org','\_seq'))

# Add calculation to subtract revenue\_org from revenue\_seq

orig\_seq['diff'] = orig\_seq['revenue\_seq'] - orig\_seq['revenue\_org']

# Select the title\_org, title\_seq, and diff

titles\_diff = orig\_seq[\_\_\_\_]

Sort by titles\_diff by diff in descending order and print the first few rows.

# Merge sequels and financials on index id

sequels\_fin = sequels.merge(financials, on='id', how='left')

# Self merge with suffixes as inner join with left on sequel and right on id

orig\_seq = sequels\_fin.merge(sequels\_fin, how='inner', left\_on='sequel',

                             right\_on='id', right\_index=True,

                             suffixes=('\_org','\_seq'))

# Add calculation to subtract revenue\_org from revenue\_seq

orig\_seq['diff'] = orig\_seq['revenue\_seq'] - orig\_seq['revenue\_org']

# Select the title\_org, title\_seq, and diff

titles\_diff = orig\_seq[['title\_org','title\_seq','diff']]

# Print the first rows of the sorted titles\_diff

print(titles\_diff.sort\_values(\_\_\_\_).head())

# Merge sequels and financials on index id sequels\_fin = sequels.merge(financials, on='id', how='left') # Self merge with suffixes as inner join with left on sequel and right on id orig\_seq = sequels\_fin.merge(sequels\_fin, how='inner', left\_on='sequel', right\_on='id', right\_index=True, suffixes=('\_org','\_seq')) # Add calculation to subtract revenue\_org from revenue\_seq orig\_seq['diff'] = orig\_seq['revenue\_seq'] - orig\_seq['revenue\_org'] # Select the title\_org, title\_seq, and diff titles\_diff = orig\_seq[['title\_org', 'title\_seq','diff']]

# Merge sequels and financials on index id

sequels\_fin = sequels.merge(financials, on='id', how='left')

# Self merge with suffixes as inner join with left on sequel and right on id

orig\_seq = sequels\_fin.merge(sequels\_fin, how='inner', left\_on='sequel',

                             right\_on='id', right\_index=True,

                             suffixes=('\_org','\_seq'))

# Add calculation to subtract revenue\_org from revenue\_seq

orig\_seq['diff'] = orig\_seq['revenue\_seq'] - orig\_seq['revenue\_org']

# Select the title\_org, title\_seq, and diff

titles\_diff = orig\_seq[['title\_org','title\_seq','diff']]

# Print the first rows of the sorted titles\_diff

print(titles\_diff.sort\_values('diff').head())

# Merge sequels and financials on index id

sequels\_fin = sequels.merge(financials, on='id', how='left')

# Self merge with suffixes as inner join with left on sequel and right on id

orig\_seq = sequels\_fin.merge(sequels\_fin, how='inner', left\_on='sequel',

right\_on='id', right\_index=True,

suffixes=('\_org','\_seq'))

# Add calculation to subtract revenue\_org from revenue\_seq

orig\_seq['diff'] = orig\_seq['revenue\_seq'] - orig\_seq['revenue\_org']

# Select the title\_org, title\_seq, and diff

titles\_diff = orig\_seq[['title\_org','title\_seq','diff']]

# Print the first rows of the sorted titles\_diff

print(titles\_diff.sort\_values('diff').head())

title\_org title\_seq diff

id

764 The Evil Dead Evil Dead II -2.348e+07

817 Austin Powers: The Spy Who Shagged Me Austin Powers in Goldmember -1.428e+07

64688 21 Jump Street 22 Jump Street -1.314e+07

36557 Casino Royale Quantum of Solace -1.296e+07

80 Before Sunset Before Midnight -4.816e+06

# Merge sequels and financials on index id

sequels\_fin = sequels.merge(financials, on='id', how='left')

# Self merge with suffixes as inner join with left on sequel and right on id

orig\_seq = sequels\_fin.merge(sequels\_fin, how='inner', left\_on='sequel',

                             right\_on='id', right\_index=True,

                             suffixes=('\_org','\_seq'))

# Add calculation to subtract revenue\_org from revenue\_seq

orig\_seq['diff'] = orig\_seq['revenue\_seq'] - orig\_seq['revenue\_org']

# Select the title\_org, title\_seq, and diff

titles\_diff = orig\_seq[['title\_org','title\_seq','diff']]

# Print the first rows of the sorted titles\_diff

print(titles\_diff.sort\_values('diff', ascending=False).head())

# Merge sequels and financials on index id

sequels\_fin = sequels.merge(financials, on='id', how='left')

# Self merge with suffixes as inner join with left on sequel and right on id

orig\_seq = sequels\_fin.merge(sequels\_fin, how='inner', left\_on='sequel',

right\_on='id', right\_index=True,

suffixes=('\_org','\_seq'))

# Add calculation to subtract revenue\_org from revenue\_seq

orig\_seq['diff'] = orig\_seq['revenue\_seq'] - orig\_seq['revenue\_org']

# Select the title\_org, title\_seq, and diff

titles\_diff = orig\_seq[['title\_org','title\_seq','diff']]

# Print the first rows of the sorted titles\_diff

print(titles\_diff.sort\_values('diff', ascending=False).head())

title\_org title\_seq diff

id

331 Jurassic Park III Jurassic World 1.145e+09

272 Batman Begins The Dark Knight 6.303e+08

10138 Iron Man 2 Iron Man 3 5.915e+08

863 Toy Story 2 Toy Story 3 5.696e+08

10764 Quantum of Solace Skyfall 5.225e+08

you needed to merge tables on their index and merge another table to itself. After the calculations were added and sub-select specific columns, the data was sorted. You found out that Jurassic World had one of the highest of all, improvement in revenue compared to the original movie.

**Daily XP250**

# Filtering joins

**50 XP**

## 1. Filtering joins

Welcome to the third chapter! In this lesson, we will discuss a type of join called a filtering join. pandas doesn't provide direct support for filtering joins, but we will learn how to replicate them.

## 2. Mutating versus filtering joins

So far, we have only worked with mutating joins, which combines data from two tables. However, filtering joins filter observations from one table based on whether or not they match an observation in another table.

## 3. What is a semi join?

Let's start with a semi join. A semi join filters the left table down to those observations that have a match in the right table. It is similar to an inner join where only the intersection between the tables is returned, but unlike an inner join, only the columns from the left table are shown. Finally, no duplicate rows from the left table are returned, even if there is a one-to-many relationship. Let's look at an example.

## 4. Musical dataset

For this chapter, the dataset we will use is from an online music streaming service.

1. 1 Photo by Vlad Bagacian from Pexels

## 5. Example datasets

In this new dataset, we have a table of song genres shown here. There's also a table of top-rated song tracks. The 'gid' column connects the two tables. Let's say we want to find what genres appear in our table of top songs. A semi join would return only the columns from the genre table and not the tracks.

## 6. Step 1 - semi join

First, let's merge the two tables with an inner join. We also print the first few rows of the genres\_tracks variable. Since this is an inner join, the returned 'gid' column holds only values where both tables matched.

## 7. Step 2 - semi join

For the next step in the technique, let's focus on this line of code. It uses a method called isin(), which compares every 'gid' in the genres table to the 'gid' in the genres\_tracks table. This will tell us if our genre appears in our merged genres\_tracks table.

## 8. Step 2 - semi join

This line of code returns a Boolean Series of true or false values.

## 9. Step 3 - semi join

To combine everything, we use that line of code to subset the genres table. The results are saved to top\_genres and we print a few rows. We've completed a semi join. These are rows in the genre table that are also found in the top\_tracks table. This is called a filtering join because we've filtered the genres table by what's in the top\_tracks table.

## 10. What is an anti join?

Now let's talk about anti joins. An anti join returns the observations in the left table that do not have a matching observation in the right table. It also only returns the columns from the left table. Now, let's go back to our example. Instead of finding which genres are in the table of top tracks, let's now find which genres are not with an anti join.

## 11. Step 1 - anti join

The first step is to use a left join returning all of the rows from the left table. Here we'll use the indicator argument and set it to True. With indicator set to True, the merge method adds a column called "\_merge" to the output. This column tells the source of each row. For example, the first four rows found a match in both tables, whereas the last can only be found in the left table.

## 12. Step 2 - anti join

Next, we use the "loc" accessor and "\_merge" column to select the rows that only appeared in the left table and return only the "gid" column from the genres\_tracks table. We now have a list of gids not in the tracks table.

## 13. Step 3 - anti join

In our final step we use the isin() method to filter for the rows with gids in our gid\_list. Our output shows those genres not in the tracks table.

## 14. Let's practice!

Now, your turn.

**Daily XP300**

**Exercise**

**Steps of a semi join**

In the last video, you were shown how to perform a semi join with pandas. In this exercise, you'll solidify your understanding of the necessary steps. Recall that a semi join filters the left table to only the rows where a match exists in both the left and right tables.

**Instructions**

**100XP**

* Sort the steps in the correct order of the technique shown to perform a semi join in pandas.
* +100 XP
* Congratulations! You have a sense of the steps in this technique. It first merges the tables, then searches it for which rows belong in the final result creating a filter and subsets the left table with that filter.

Step 1: Merge the 2 tables left and right on key columns using inner join

Step 2: Search key column in left table .isin() method in the merged tables using this method creating a Boolean series

Step 3: Subset the rows of left table

+100 XP

Congratulations! You have a sense of the steps in this technique. It first merges the tables, then searches it for which rows belong in the final result creating a filter and subsets the left table with that filter.

**Daily XP400**

**Exercise**

**Exercise**

**Performing an anti join**

In our music streaming company dataset, each customer is assigned an employee representative to assist them. In this exercise, filter the employee table by a table of top customers, returning only those employees who are **not** assigned to a customer. The results should resemble the results of an anti join. The company's leadership will assign these employees additional training so that they can work with high valued customers.

The top\_cust and employees tables have been provided for you.

**Instructions 1/3**

**35 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))
* Merge employees and top\_cust with a left join, setting indicator argument to True. Save the result to empl\_cust.
* # Merge employees and top\_cust
* empl\_cust = \_\_\_\_.merge(\_\_\_\_, on=\_\_\_\_,
* how=\_\_\_\_, indicator=\_\_\_\_)

# Merge employees and top\_cust

empl\_cust = employees.merge(top\_cust, on='srid',

                            how='left', indicator=True)

print(empl\_cust)

In [1]:

print(employees)

srid lname fname title hire\_date email

0 1 Adams Andrew General Manager 2002-08-14 andrew@chinookcorp.com

1 2 Edwards Nancy Sales Manager 2002-05-01 nancy@chinookcorp.com

2 3 Peacock Jane Sales Support Agent 2002-04-01 jane@chinookcorp.com

3 4 Park Margaret Sales Support Agent 2003-05-03 margaret@chinookcorp.com

4 5 Johnson Steve Sales Support Agent 2003-10-17 steve@chinookcorp.com

5 6 Mitchell Michael IT Manager 2003-10-17 michael@chinookcorp.com

6 7 King Robert IT Staff 2004-01-02 robert@chinookcorp.com

7 8 Callahan Laura IT Staff 2004-03-04 laura@chinookcorp.com

In [2]:

print(top\_cust)

cid srid fname lname phone fax email

0 1 3 Luís Gonçalves +55 (12) 3923-5555 +55 (12) 3923-5566 luisg@embraer.com.br

1 2 5 Leonie Köhler +49 0711 2842222 NaN leonekohler@surfeu.de

2 3 3 François Tremblay +1 (514) 721-4711 NaN ftremblay@gmail.com

3 4 4 Bjørn Hansen +47 22 44 22 22 NaN bjorn.hansen@yahoo.no

4 5 4 František Wichterlová +420 2 4172 5555 +420 2 4172 5555 frantisekw@jetbrains.com

5 6 5 Helena Holý +420 2 4177 0449 NaN hholy@gmail.com

6 7 5 Astrid Gruber +43 01 5134505 NaN astrid.gruber@apple.at

7 8 4 Daan Peeters +32 02 219 03 03 NaN daan\_peeters@apple.be

8 9 4 Kara Nielsen +453 3331 9991 NaN kara.nielsen@jubii.dk

9 10 4 Eduardo Martins +55 (11) 3033-5446 +55 (11) 3033-4564 eduardo@woodstock.com.br

10 11 5 Alexandre Rocha +55 (11) 3055-3278 +55 (11) 3055-8131 alero@uol.com.br

11 12 3 Roberto Almeida +55 (21) 2271-7000 +55 (21) 2271-7070 roberto.almeida@riotur.gov.br

12 13 4 Fernanda Ramos +55 (61) 3363-5547 +55 (61) 3363-7855 fernadaramos4@uol.com.br

13 14 5 Mark Philips +1 (780) 434-4554 +1 (780) 434-5565 mphilips12@shaw.ca

14 15 3 Jennifer Peterson +1 (604) 688-2255 +1 (604) 688-8756 jenniferp@rogers.ca

15 16 4 Frank Harris +1 (650) 253-0000 +1 (650) 253-0000 fharris@google.com

16 17 5 Jack Smith +1 (425) 882-8080 +1 (425) 882-8081 jacksmith@microsoft.com

17 18 3 Michelle Brooks +1 (212) 221-3546 +1 (212) 221-4679 michelleb@aol.com

18 19 3 Tim Goyer +1 (408) 996-1010 +1 (408) 996-1011 tgoyer@apple.com

19 20 4 Dan Miller +1 (650) 644-3358 NaN dmiller@comcast.com

20 21 5 Kathy Chase +1 (775) 223-7665 NaN kachase@hotmail.com

21 22 4 Heather Leacock +1 (407) 999-7788 NaN hleacock@gmail.com

22 23 4 John Gordon +1 (617) 522-1333 NaN johngordon22@yahoo.com

23 24 3 Frank Ralston +1 (312) 332-3232 NaN fralston@gmail.com

24 25 5 Victor Stevens +1 (608) 257-0597 NaN vstevens@yahoo.com

25 26 4 Richard Cunningham +1 (817) 924-7272 NaN ricunningham@hotmail.com

26 27 4 Patrick Gray +1 (520) 622-4200 NaN patrick.gray@aol.com

27 28 5 Julia Barnett +1 (801) 531-7272 NaN jubarnett@gmail.com

28 29 3 Robert Brown +1 (416) 363-8888 NaN robbrown@shaw.ca

29 30 3 Edward Francis +1 (613) 234-3322 NaN edfrancis@yachoo.ca

30 31 5 Martha Silk +1 (902) 450-0450 NaN marthasilk@gmail.com

31 32 4 Aaron Mitchell +1 (204) 452-6452 NaN aaronmitchell@yahoo.ca

32 33 3 Ellie Sullivan +1 (867) 920-2233 NaN ellie.sullivan@shaw.ca

33 34 4 João Fernandes +351 (213) 466-111 NaN jfernandes@yahoo.pt

34 35 4 Madalena Sampaio +351 (225) 022-448 NaN masampaio@sapo.pt

35 36 5 Hannah Schneider +49 030 26550280 NaN hannah.schneider@yahoo.de

36 37 3 Fynn Zimmermann +49 069 40598889 NaN fzimmermann@yahoo.de

37 38 3 Niklas Schröder +49 030 2141444 NaN nschroder@surfeu.de

38 39 4 Camille Bernard +33 01 49 70 65 65 NaN camille.bernard@yahoo.fr

39 40 4 Dominique Lefebvre +33 01 47 42 71 71 NaN dominiquelefebvre@gmail.com

40 41 5 Marc Dubois +33 04 78 30 30 30 NaN marc.dubois@hotmail.com

41 42 3 Wyatt Girard +33 05 56 96 96 96 NaN wyatt.girard@yahoo.fr

42 43 3 Isabelle Mercier +33 03 80 73 66 99 NaN isabelle\_mercier@apple.fr

43 44 3 Terhi Hämäläinen +358 09 870 2000 NaN terhi.hamalainen@apple.fi

44 45 3 Ladislav Kovács NaN NaN ladislav\_kovacs@apple.hu

45 46 3 Hugh O'Reilly +353 01 6792424 NaN hughoreilly@apple.ie

46 47 5 Lucas Mancini +39 06 39733434 NaN lucas.mancini@yahoo.it

47 48 5 Johannes Van der Berg +31 020 6223130 NaN johavanderberg@yahoo.nl

48 49 4 Stanisław Wójcik +48 22 828 37 39 NaN stanisław.wójcik@wp.pl

49 50 5 Enrique Muñoz +34 914 454 454 NaN enrique\_munoz@yahoo.es

50 51 5 Joakim Johansson +46 08-651 52 52 NaN joakim.johansson@yahoo.se

51 52 3 Emma Jones +44 020 7707 0707 NaN emma\_jones@hotmail.com

52 53 3 Phil Hughes +44 020 7976 5722 NaN phil.hughes@gmail.com

53 54 5 Steve Murray +44 0131 315 3300 NaN steve.murray@yahoo.uk

54 55 4 Mark Taylor +61 (02) 9332 3633 NaN mark.taylor@yahoo.au

55 56 4 Diego Gutiérrez +54 (0)11 4311 4333 NaN diego.gutierrez@yahoo.ar

56 57 5 Luis Rojas +56 (0)2 635 4444 NaN luisrojas@yahoo.cl

57 58 3 Manoj Pareek +91 0124 39883988 NaN manoj.pareek@rediff.com

58 59 3 Puja Srivastava +91 080 22289999 NaN puja\_srivastava@yahoo.in

# Merge employees and top\_cust

empl\_cust = employees.merge(top\_cust, on='srid',

how='left', indicator=True)

print(empl\_cust)

srid lname\_x fname\_x title hire\_date ... lname\_y phone fax email\_y \_merge

0 1 Adams Andrew General Manager 2002-08-14 ... NaN NaN NaN NaN left\_only

1 2 Edwards Nancy Sales Manager 2002-05-01 ... NaN NaN NaN NaN left\_only

2 3 Peacock Jane Sales Support Agent 2002-04-01 ... Gonçalves +55 (12) 3923-5555 +55 (12) 3923-5566 luisg@embraer.com.br both

3 3 Peacock Jane Sales Support Agent 2002-04-01 ... Tremblay +1 (514) 721-4711 NaN ftremblay@gmail.com both

4 3 Peacock Jane Sales Support Agent 2002-04-01 ... Almeida +55 (21) 2271-7000 +55 (21) 2271-7070 roberto.almeida@riotur.gov.br both

.. ... ... ... ... ... ... ... ... ... ... ...

59 5 Johnson Steve Sales Support Agent 2003-10-17 ... Murray +44 0131 315 3300 NaN steve.murray@yahoo.uk both

60 5 Johnson Steve Sales Support Agent 2003-10-17 ... Rojas +56 (0)2 635 4444 NaN luisrojas@yahoo.cl both

61 6 Mitchell Michael IT Manager 2003-10-17 ... NaN NaN NaN NaN left\_only

62 7 King Robert IT Staff 2004-01-02 ... NaN NaN NaN NaN left\_only

63 8 Callahan Laura IT Staff 2004-03-04 ... NaN NaN NaN NaN left\_only

[64 rows x 13 columns]

In [1]:

print(employees)

srid lname fname title hire\_date email

0 1 Adams Andrew General Manager 2002-08-14 andrew@chinookcorp.com

1 2 Edwards Nancy Sales Manager 2002-05-01 nancy@chinookcorp.com

2 3 Peacock Jane Sales Support Agent 2002-04-01 jane@chinookcorp.com

3 4 Park Margaret Sales Support Agent 2003-05-03 margaret@chinookcorp.com

4 5 Johnson Steve Sales Support Agent 2003-10-17 steve@chinookcorp.com

5 6 Mitchell Michael IT Manager 2003-10-17 michael@chinookcorp.com

6 7 King Robert IT Staff 2004-01-02 robert@chinookcorp.com

7 8 Callahan Laura IT Staff 2004-03-04 laura@chinookcorp.com

In [2]:

print(top\_cust)

cid srid fname lname phone fax email

0 1 3 Luís Gonçalves +55 (12) 3923-5555 +55 (12) 3923-5566 luisg@embraer.com.br

1 2 5 Leonie Köhler +49 0711 2842222 NaN leonekohler@surfeu.de

2 3 3 François Tremblay +1 (514) 721-4711 NaN ftremblay@gmail.com

3 4 4 Bjørn Hansen +47 22 44 22 22 NaN bjorn.hansen@yahoo.no

4 5 4 František Wichterlová +420 2 4172 5555 +420 2 4172 5555 frantisekw@jetbrains.com

5 6 5 Helena Holý +420 2 4177 0449 NaN hholy@gmail.com

6 7 5 Astrid Gruber +43 01 5134505 NaN astrid.gruber@apple.at

7 8 4 Daan Peeters +32 02 219 03 03 NaN daan\_peeters@apple.be

8 9 4 Kara Nielsen +453 3331 9991 NaN kara.nielsen@jubii.dk

9 10 4 Eduardo Martins +55 (11) 3033-5446 +55 (11) 3033-4564 eduardo@woodstock.com.br

10 11 5 Alexandre Rocha +55 (11) 3055-3278 +55 (11) 3055-8131 alero@uol.com.br

11 12 3 Roberto Almeida +55 (21) 2271-7000 +55 (21) 2271-7070 roberto.almeida@riotur.gov.br

12 13 4 Fernanda Ramos +55 (61) 3363-5547 +55 (61) 3363-7855 fernadaramos4@uol.com.br

13 14 5 Mark Philips +1 (780) 434-4554 +1 (780) 434-5565 mphilips12@shaw.ca

14 15 3 Jennifer Peterson +1 (604) 688-2255 +1 (604) 688-8756 jenniferp@rogers.ca

15 16 4 Frank Harris +1 (650) 253-0000 +1 (650) 253-0000 fharris@google.com

16 17 5 Jack Smith +1 (425) 882-8080 +1 (425) 882-8081 jacksmith@microsoft.com

17 18 3 Michelle Brooks +1 (212) 221-3546 +1 (212) 221-4679 michelleb@aol.com

18 19 3 Tim Goyer +1 (408) 996-1010 +1 (408) 996-1011 tgoyer@apple.com

19 20 4 Dan Miller +1 (650) 644-3358 NaN dmiller@comcast.com

20 21 5 Kathy Chase +1 (775) 223-7665 NaN kachase@hotmail.com

21 22 4 Heather Leacock +1 (407) 999-7788 NaN hleacock@gmail.com

22 23 4 John Gordon +1 (617) 522-1333 NaN johngordon22@yahoo.com

23 24 3 Frank Ralston +1 (312) 332-3232 NaN fralston@gmail.com

24 25 5 Victor Stevens +1 (608) 257-0597 NaN vstevens@yahoo.com

25 26 4 Richard Cunningham +1 (817) 924-7272 NaN ricunningham@hotmail.com

26 27 4 Patrick Gray +1 (520) 622-4200 NaN patrick.gray@aol.com

27 28 5 Julia Barnett +1 (801) 531-7272 NaN jubarnett@gmail.com

28 29 3 Robert Brown +1 (416) 363-8888 NaN robbrown@shaw.ca

29 30 3 Edward Francis +1 (613) 234-3322 NaN edfrancis@yachoo.ca

30 31 5 Martha Silk +1 (902) 450-0450 NaN marthasilk@gmail.com

31 32 4 Aaron Mitchell +1 (204) 452-6452 NaN aaronmitchell@yahoo.ca

32 33 3 Ellie Sullivan +1 (867) 920-2233 NaN ellie.sullivan@shaw.ca

33 34 4 João Fernandes +351 (213) 466-111 NaN jfernandes@yahoo.pt

34 35 4 Madalena Sampaio +351 (225) 022-448 NaN masampaio@sapo.pt

35 36 5 Hannah Schneider +49 030 26550280 NaN hannah.schneider@yahoo.de

36 37 3 Fynn Zimmermann +49 069 40598889 NaN fzimmermann@yahoo.de

37 38 3 Niklas Schröder +49 030 2141444 NaN nschroder@surfeu.de

38 39 4 Camille Bernard +33 01 49 70 65 65 NaN camille.bernard@yahoo.fr

39 40 4 Dominique Lefebvre +33 01 47 42 71 71 NaN dominiquelefebvre@gmail.com

40 41 5 Marc Dubois +33 04 78 30 30 30 NaN marc.dubois@hotmail.com

41 42 3 Wyatt Girard +33 05 56 96 96 96 NaN wyatt.girard@yahoo.fr

42 43 3 Isabelle Mercier +33 03 80 73 66 99 NaN isabelle\_mercier@apple.fr

43 44 3 Terhi Hämäläinen +358 09 870 2000 NaN terhi.hamalainen@apple.fi

44 45 3 Ladislav Kovács NaN NaN ladislav\_kovacs@apple.hu

45 46 3 Hugh O'Reilly +353 01 6792424 NaN hughoreilly@apple.ie

46 47 5 Lucas Mancini +39 06 39733434 NaN lucas.mancini@yahoo.it

47 48 5 Johannes Van der Berg +31 020 6223130 NaN johavanderberg@yahoo.nl

48 49 4 Stanisław Wójcik +48 22 828 37 39 NaN stanisław.wójcik@wp.pl

49 50 5 Enrique Muñoz +34 914 454 454 NaN enrique\_munoz@yahoo.es

50 51 5 Joakim Johansson +46 08-651 52 52 NaN joakim.johansson@yahoo.se

51 52 3 Emma Jones +44 020 7707 0707 NaN emma\_jones@hotmail.com

52 53 3 Phil Hughes +44 020 7976 5722 NaN phil.hughes@gmail.com

53 54 5 Steve Murray +44 0131 315 3300 NaN steve.murray@yahoo.uk

54 55 4 Mark Taylor +61 (02) 9332 3633 NaN mark.taylor@yahoo.au

55 56 4 Diego Gutiérrez +54 (0)11 4311 4333 NaN diego.gutierrez@yahoo.ar

56 57 5 Luis Rojas +56 (0)2 635 4444 NaN luisrojas@yahoo.cl

57 58 3 Manoj Pareek +91 0124 39883988 NaN manoj.pareek@rediff.com

58 59 3 Puja Srivastava +91 080 22289999 NaN puja\_srivastava@yahoo.in

# Merge employees and top\_cust

empl\_cust = employees.merge(top\_cust, on='srid',

how='left', indicator=True)

print(empl\_cust)

srid lname\_x fname\_x title hire\_date ... lname\_y phone fax email\_y \_merge

0 1 Adams Andrew General Manager 2002-08-14 ... NaN NaN NaN NaN left\_only

1 2 Edwards Nancy Sales Manager 2002-05-01 ... NaN NaN NaN NaN left\_only

2 3 Peacock Jane Sales Support Agent 2002-04-01 ... Gonçalves +55 (12) 3923-5555 +55 (12) 3923-5566 luisg@embraer.com.br both

3 3 Peacock Jane Sales Support Agent 2002-04-01 ... Tremblay +1 (514) 721-4711 NaN ftremblay@gmail.com both

4 3 Peacock Jane Sales Support Agent 2002-04-01 ... Almeida +55 (21) 2271-7000 +55 (21) 2271-7070 roberto.almeida@riotur.gov.br both

.. ... ... ... ... ... ... ... ... ... ... ...

59 5 Johnson Steve Sales Support Agent 2003-10-17 ... Murray +44 0131 315 3300 NaN steve.murray@yahoo.uk both

60 5 Johnson Steve Sales Support Agent 2003-10-17 ... Rojas +56 (0)2 635 4444 NaN luisrojas@yahoo.cl both

61 6 Mitchell Michael IT Manager 2003-10-17 ... NaN NaN NaN NaN left\_only

62 7 King Robert IT Staff 2004-01-02 ... NaN NaN NaN NaN left\_only

63 8 Callahan Laura IT Staff 2004-03-04 ... NaN NaN NaN NaN left\_only

[64 rows x 13 columns]

# Merge employees and top\_cust

empl\_cust = employees.merge(top\_cust, on='srid',

                            how='left', indicator=True)

# Select the srid column where \_merge is left\_only

srid\_list = empl\_cust.loc[\_\_\_\_, 'srid']

# Merge employees and top\_cust

empl\_cust = employees.merge(top\_cust, on='srid',

                            how='left', indicator=True)

# Select the srid column where \_merge is left\_only

srid\_list = empl\_cust.loc[empl\_cust['\_merge'] =='left\_only', 'srid']

print(srid\_list)

# Merge employees and top\_cust

empl\_cust = employees.merge(top\_cust, on='srid',

how='left', indicator=True)

# Select the srid column where \_merge is left\_only

srid\_list = empl\_cust.loc[empl\_cust['\_merge'] =='left\_only', 'srid']

print(srid\_list)

0 1

1 2

61 6

62 7

63 8

Name: srid, dtype: int64

# Merge employees and top\_cust

empl\_cust = employees.merge(top\_cust, on='srid',

how='left', indicator=True)

# Select the srid column where \_merge is left\_only

srid\_list = empl\_cust.loc[empl\_cust['\_merge'] =='left\_only', 'srid']

print(srid\_list)

0 1

1 2

61 6

62 7

63 8

Name: srid, dtype: int64

# Merge employees and top\_cust

empl\_cust = employees.merge(top\_cust, on='srid',

                                 how='left', indicator=True)

# Select the srid column where \_merge is left\_only

srid\_list = empl\_cust.loc[empl\_cust['\_merge'] == 'left\_only', 'srid']

# Get employees not working with top customers

print(employees[\_\_\_\_.isin(\_\_\_\_)])

# Merge employees and top\_cust

empl\_cust = employees.merge(top\_cust, on='srid',

how='left', indicator=True)

# Select the srid column where \_merge is left\_only

srid\_list = empl\_cust.loc[empl\_cust['\_merge'] =='left\_only', 'srid']

print(srid\_list)

0 1

1 2

61 6

62 7

63 8

Name: srid, dtype: int64

<script.py> output:

0 1

1 2

61 6

62 7

63 8

Name: srid, dtype: int64

# Merge employees and top\_cust

empl\_cust = employees.merge(top\_cust, on='srid',

                                 how='left', indicator=True)

# Select the srid column where \_merge is left\_only

srid\_list = empl\_cust.loc[empl\_cust['\_merge'] == 'left\_only', 'srid']

# Get employees not working with top customers

print(employees[employees['srid'].isin(srid\_list)])

# Merge employees and top\_cust

empl\_cust = employees.merge(top\_cust, on='srid',

how='left', indicator=True)

# Select the srid column where \_merge is left\_only

srid\_list = empl\_cust.loc[empl\_cust['\_merge'] == 'left\_only', 'srid']

# Get employees not working with top customers

print(employees[employees['srid'].isin(srid\_list)])

srid lname fname title hire\_date email

0 1 Adams Andrew General Manager 2002-08-14 andrew@chinookcorp.com

1 2 Edwards Nancy Sales Manager 2002-05-01 nancy@chinookcorp.com

5 6 Mitchell Michael IT Manager 2003-10-17 michael@chinookcorp.com

6 7 King Robert IT Staff 2004-01-02 robert@chinookcorp.com

7 8 Callahan Laura IT Staff 2004-03-04 laura@chinookcorp.com

Success! You performed an anti join by first merging the tables with a left join, selecting the ID of those employees who did not support a top customer, and then subsetting the original employee's table. From that, we can see that there are five employees not supporting top customers. Anti joins are a powerful tool to filter a main table (i.e. employees) by another (i.e. customers).

# Merge employees and top\_cust

empl\_cust = employees.merge(top\_cust, on='srid',

                                 how='left', indicator=True)

# Select the srid column where \_merge is left\_only

srid\_list = empl\_cust.loc[empl\_cust['\_merge'] == 'left\_only', 'srid']

# Get employees not working with top customers

print(employees[employees['srid'].isin(srid\_list)])

# Merge employees and top\_cust

empl\_cust = employees.merge(top\_cust, on='srid',

how='left', indicator=True)

# Select the srid column where \_merge is left\_only

srid\_list = empl\_cust.loc[empl\_cust['\_merge'] == 'left\_only', 'srid']

# Get employees not working with top customers

print(employees[employees['srid'].isin(srid\_list)])

srid lname fname title hire\_date email

0 1 Adams Andrew General Manager 2002-08-14 andrew@chinookcorp.com

1 2 Edwards Nancy Sales Manager 2002-05-01 nancy@chinookcorp.com

5 6 Mitchell Michael IT Manager 2003-10-17 michael@chinookcorp.com

6 7 King Robert IT Staff 2004-01-02 robert@chinookcorp.com

7 8 Callahan Laura IT Staff 2004-03-04 laura@chinookcorp.com

<script.py> output:

srid lname fname title hire\_date email

0 1 Adams Andrew General Manager 2002-08-14 andrew@chinookcorp.com

1 2 Edwards Nancy Sales Manager 2002-05-01 nancy@chinookcorp.com

5 6 Mitchell Michael IT Manager 2003-10-17 michael@chinookcorp.com

6 7 King Robert IT Staff 2004-01-02 robert@chinookcorp.com

7 8 Callahan Laura IT Staff 2004-03-04 laura@chinookcorp.com

**Daily XP500**

**Exercise**

**Exercise**

**Performing a semi join**

Some of the tracks that have generated the most significant amount of revenue are from TV-shows or are other non-musical audio. You have been given a table of invoices that include top revenue-generating items. Additionally, you have a table of non-musical tracks from the streaming service. In this exercise, you'll use a semi join to find the top revenue-generating non-musical tracks..

The tables non\_mus\_tcks, top\_invoices, and genres have been loaded for you.

**Instructions**

**100 XP**

* Merge non\_mus\_tcks and top\_invoices on tid using an inner join. Save the result as tracks\_invoices.
* Use .isin() to subset the rows of non\_mus\_tck where tid is in the tid column of tracks\_invoices. Save the result as top\_tracks.
* Group top\_tracks by gid and count the tid rows. Save the result to cnt\_by\_gid.
* Merge cnt\_by\_gid with the genres table on gid and print the result.

# Merge the non\_mus\_tck and top\_invoices tables on tid

tracks\_invoices = \_\_\_\_.merge(\_\_\_\_)

# Use .isin() to subset non\_mus\_tcks to rows with tid in tracks\_invoices

top\_tracks = \_\_\_\_\_[non\_mus\_tcks['tid'].isin(\_\_\_\_)]

# Group the top\_tracks by gid and count the tid rows

cnt\_by\_gid = top\_tracks.groupby(['gid'], as\_index=False).agg({'tid':\_\_\_\_})

# Merge the genres table to cnt\_by\_gid on gid and print

print(\_\_\_\_)

# Merge employees and top\_cust empl\_cust = employees.merge(top\_cust, on='srid', how='left', indicator=True) # Select the srid column where \_merge is left\_only srid\_list = empl\_cust.loc[empl\_cust['\_merge'] == 'left\_only', 'srid'] # Get employees not working with top customers print(employees[employees['srid'].isin(srid\_list)]

# Merge the non\_mus\_tck and top\_invoices tables on tid

tracks\_invoices = non\_mus\_tcks.merge(top\_invoices, on='tid', how='inner')

print(tracks\_invoices)

# Use .isin() to subset non\_mus\_tcks to rows with tid in tracks\_invoices

top\_tracks = non\_mus\_tcks[non\_mus\_tcks['tid'].isin(tracks\_invoices['tid'])]

print(top\_tracks)

# Group the top\_tracks by gid and count the tid rows

cnt\_by\_gid = top\_tracks.groupby(['gid'], as\_index=False).agg({'tid': 'count'})

# Merge the genres table to cnt\_by\_gid on gid and print

print(cnt\_by\_gid)

# Merge the non\_mus\_tck and top\_invoices tables on tid

tracks\_invoices = non\_mus\_tcks.merge(top\_invoices, on='tid', how='inner')

print(tracks\_invoices)

# Use .isin() to subset non\_mus\_tcks to rows with tid in tracks\_invoices

top\_tracks = non\_mus\_tcks[non\_mus\_tcks['tid'].isin(tracks\_invoices['tid'])]

print(top\_tracks)

# Group the top\_tracks by gid and count the tid rows

cnt\_by\_gid = top\_tracks.groupby(['gid'], as\_index=False).agg({'tid': 'count'})

# Merge the genres table to cnt\_by\_gid on gid and print

print(cnt\_by\_gid)

tid name aid mtid gid u\_price ilid iid uprice quantity

0 2850 The Fix 228 3 21 1.99 473 88 1.99 1

1 2850 The Fix 228 3 21 1.99 2192 404 1.99 1

2 2868 Walkabout 230 3 19 1.99 476 88 1.99 1

3 2868 Walkabout 230 3 19 1.99 2194 404 1.99 1

4 3177 Hot Girl 249 3 19 1.99 1668 306 1.99 1

5 3177 Hot Girl 249 3 19 1.99 2240 412 1.99 1

6 3200 Gay Witch Hunt 251 3 19 1.99 1098 201 1.99 1

7 3200 Gay Witch Hunt 251 3 19 1.99 1670 307 1.99 1

8 3214 Phyllis's Wedding 251 3 22 1.99 527 96 1.99 1

9 3214 Phyllis's Wedding 251 3 22 1.99 1678 310 1.99 1

10 3223 How to Stop an Exploding Man 228 3 21 1.99 528 96 1.99 1

11 3223 How to Stop an Exploding Man 228 3 21 1.99 1100 202 1.99 1

12 3250 Pilot 254 3 19 1.99 533 99 1.99 1

13 3250 Pilot 254 3 19 1.99 1686 312 1.99 1

tid name aid mtid gid u\_price

2849 2850 The Fix 228 3 21 1.99

2867 2868 Walkabout 230 3 19 1.99

3176 3177 Hot Girl 249 3 19 1.99

3199 3200 Gay Witch Hunt 251 3 19 1.99

3213 3214 Phyllis's Wedding 251 3 22 1.99

3222 3223 How to Stop an Exploding Man 228 3 21 1.99

3249 3250 Pilot 254 3 19 1.99

gid tid

0 19 4

1 21 2

2 22 1

# Merge employees and top\_cust

empl\_cust = employees.merge(top\_cust, on='srid',

how='left', indicator=True)

# Select the srid column where \_merge is left\_only

srid\_list = empl\_cust.loc[empl\_cust['\_merge'] == 'left\_only', 'srid']

# Get employees not working with top customers

print(employees[employees['srid'].isin(srid\_list)])

In [1]:

print(non\_mus\_tcks)

tid name aid mtid gid u\_price

2819 2820 Occupation / Precipice 227 3 19 1.99

2820 2821 Exodus, Pt. 1 227 3 19 1.99

2821 2822 Exodus, Pt. 2 227 3 19 1.99

2822 2823 Collaborators 227 3 19 1.99

2823 2824 Torn 227 3 19 1.99

... ... ... ... ... ... ...

3361 3362 There's No Place Like Home, Pt. 1 261 3 21 1.99

3362 3363 There's No Place Like Home, Pt. 2 261 3 21 1.99

3363 3364 There's No Place Like Home, Pt. 3 261 3 21 1.99

3427 3428 Branch Closing 251 3 22 1.99

3428 3429 The Return 251 3 22 1.99

[200 rows x 6 columns]

# Merge the non\_mus\_tck and top\_invoices tables on tid

tracks\_invoices = non\_mus\_tcks.merge(top\_invoices, on='tid', how='inner')

print(tracks\_invoices)

# Use .isin() to subset non\_mus\_tcks to rows with tid in tracks\_invoices

top\_tracks = non\_mus\_tcks[non\_mus\_tcks['tid'].isin(tracks\_invoices['tid'])]

print(top\_tracks)

# Group the top\_tracks by gid and count the tid rows

cnt\_by\_gid = top\_tracks.groupby(['gid'], as\_index=False).agg({'tid': 'count'})

# Merge the genres table to cnt\_by\_gid on gid and print

print(cnt\_by\_gid)

tid name aid mtid gid u\_price ilid iid uprice quantity

0 2850 The Fix 228 3 21 1.99 473 88 1.99 1

1 2850 The Fix 228 3 21 1.99 2192 404 1.99 1

2 2868 Walkabout 230 3 19 1.99 476 88 1.99 1

3 2868 Walkabout 230 3 19 1.99 2194 404 1.99 1

4 3177 Hot Girl 249 3 19 1.99 1668 306 1.99 1

5 3177 Hot Girl 249 3 19 1.99 2240 412 1.99 1

6 3200 Gay Witch Hunt 251 3 19 1.99 1098 201 1.99 1

7 3200 Gay Witch Hunt 251 3 19 1.99 1670 307 1.99 1

8 3214 Phyllis's Wedding 251 3 22 1.99 527 96 1.99 1

9 3214 Phyllis's Wedding 251 3 22 1.99 1678 310 1.99 1

10 3223 How to Stop an Exploding Man 228 3 21 1.99 528 96 1.99 1

11 3223 How to Stop an Exploding Man 228 3 21 1.99 1100 202 1.99 1

12 3250 Pilot 254 3 19 1.99 533 99 1.99 1

13 3250 Pilot 254 3 19 1.99 1686 312 1.99 1

tid name aid mtid gid u\_price

2849 2850 The Fix 228 3 21 1.99

2867 2868 Walkabout 230 3 19 1.99

3176 3177 Hot Girl 249 3 19 1.99

3199 3200 Gay Witch Hunt 251 3 19 1.99

3213 3214 Phyllis's Wedding 251 3 22 1.99

3222 3223 How to Stop an Exploding Man 228 3 21 1.99

3249 3250 Pilot 254 3 19 1.99

gid tid

0 19 4

1 21 2

2 22 1

<script.py> output:

tid name aid mtid gid u\_price ilid iid uprice quantity

0 2850 The Fix 228 3 21 1.99 473 88 1.99 1

1 2850 The Fix 228 3 21 1.99 2192 404 1.99 1

2 2868 Walkabout 230 3 19 1.99 476 88 1.99 1

3 2868 Walkabout 230 3 19 1.99 2194 404 1.99 1

4 3177 Hot Girl 249 3 19 1.99 1668 306 1.99 1

5 3177 Hot Girl 249 3 19 1.99 2240 412 1.99 1

6 3200 Gay Witch Hunt 251 3 19 1.99 1098 201 1.99 1

7 3200 Gay Witch Hunt 251 3 19 1.99 1670 307 1.99 1

8 3214 Phyllis's Wedding 251 3 22 1.99 527 96 1.99 1

9 3214 Phyllis's Wedding 251 3 22 1.99 1678 310 1.99 1

10 3223 How to Stop an Exploding Man 228 3 21 1.99 528 96 1.99 1

11 3223 How to Stop an Exploding Man 228 3 21 1.99 1100 202 1.99 1

12 3250 Pilot 254 3 19 1.99 533 99 1.99 1

13 3250 Pilot 254 3 19 1.99 1686 312 1.99 1

tid name aid mtid gid u\_price

2849 2850 The Fix 228 3 21 1.99

2867 2868 Walkabout 230 3 19 1.99

3176 3177 Hot Girl 249 3 19 1.99

3199 3200 Gay Witch Hunt 251 3 19 1.99

3213 3214 Phyllis's Wedding 251 3 22 1.99

3222 3223 How to Stop an Exploding Man 228 3 21 1.99

3249 3250 Pilot 254 3 19 1.99

gid tid

0 19 4

1 21 2

2 22 1

# Merge the non\_mus\_tck and top\_invoices tables on tid

tracks\_invoices = non\_mus\_tcks.merge(top\_invoices, on='tid', how='inner')

print(tracks\_invoices)

# Use .isin() to subset non\_mus\_tcks to rows with tid in tracks\_invoices

top\_tracks = non\_mus\_tcks[non\_mus\_tcks['tid'].isin(tracks\_invoices['tid'])]

print(top\_tracks)

# Group the top\_tracks by gid and count the tid rows

cnt\_by\_gid = top\_tracks.groupby(['gid'], as\_index=False).agg({'tid': 'count'})

# Merge the genres table to cnt\_by\_gid on gid and print

print(cnt\_by\_gid.merge(genres, on='gid'))

tid name aid mtid gid u\_price ilid iid uprice quantity

0 2850 The Fix 228 3 21 1.99 473 88 1.99 1

1 2850 The Fix 228 3 21 1.99 2192 404 1.99 1

2 2868 Walkabout 230 3 19 1.99 476 88 1.99 1

3 2868 Walkabout 230 3 19 1.99 2194 404 1.99 1

4 3177 Hot Girl 249 3 19 1.99 1668 306 1.99 1

5 3177 Hot Girl 249 3 19 1.99 2240 412 1.99 1

6 3200 Gay Witch Hunt 251 3 19 1.99 1098 201 1.99 1

7 3200 Gay Witch Hunt 251 3 19 1.99 1670 307 1.99 1

8 3214 Phyllis's Wedding 251 3 22 1.99 527 96 1.99 1

9 3214 Phyllis's Wedding 251 3 22 1.99 1678 310 1.99 1

10 3223 How to Stop an Exploding Man 228 3 21 1.99 528 96 1.99 1

11 3223 How to Stop an Exploding Man 228 3 21 1.99 1100 202 1.99 1

12 3250 Pilot 254 3 19 1.99 533 99 1.99 1

13 3250 Pilot 254 3 19 1.99 1686 312 1.99 1

tid name aid mtid gid u\_price

2849 2850 The Fix 228 3 21 1.99

2867 2868 Walkabout 230 3 19 1.99

3176 3177 Hot Girl 249 3 19 1.99

3199 3200 Gay Witch Hunt 251 3 19 1.99

3213 3214 Phyllis's Wedding 251 3 22 1.99

3222 3223 How to Stop an Exploding Man 228 3 21 1.99

3249 3250 Pilot 254 3 19 1.99

gid tid name

0 19 4 TV Shows

1 21 2 Drama

2 22 1 Comedy

Nice job! In this exercise, you replicated a semi join to filter the table of tracks by the table of invoice items to find the top revenue non-musical tracks. With some additional data manipulation, you discovered that \_'TV-shows'\_ is the non-musical genre that has the most top revenue-generating tracks. Now that you've done both semi- and anti joins, it's time to move to the next topic.

**Daily XP600**

# Concatenate DataFrames together vertically

**50 XP**

## 1. Concatenate DataFrames together vertically

Hello there! In this lesson, we'll talk about how to connect two tables vertically.

## 2. Concatenate two tables vertically

So far in this course, we have only discussed how to merge two tables, which mainly grows them horizontally. But what if we wanted to grow them vertically? We can use the concat method to concatenate, or stick tables together, vertically or horizontally, but in this lesson, we'll focus on vertical concatenation.

## 3. Basic concatenation

Often, data for different periods of time will come in multiple tables, but if we want to analyze it together, we'll need to combine them into one. Here are three separate tables of invoice data from our streaming service. Notice the column headers are the same. The separate tables are named "inv" underscore Jan through March.

## 4. Basic concatenation

We can pass a list of table names into pandas dot concat to combine the tables in the order they're passed in. To concatenate vertically, the axis argument should be set to 0, but 0 is the default, so we don't need to explicitly write this. The result is a vertically combined table. Notice each table's index value was retained.

## 5. Ignoring the index

If the index contains no valuable information, then we can ignore it in the concat method by setting ignore\_index to True. The result is that the index will go from 0 to n-1.

## 6. Setting labels to original tables

Now, suppose we wanted to associate specific keys with each of the pieces of our three original tables. We can provide a list of labels to the keys argument. Make sure that ignore\_index argument is False, since you can't add a key and ignore the index at the same time. This results in a table with a multi-index, with the label on the first level.

## 7. Concatenate tables with different column names

What if we need to combine tables that have different column names? The "inv\_feb" table now has a column added for billing country.

## 8. Concatenate tables with different column names

The concat method by default will include all of the columns in the different tables it's combining. The sort argument, if true, will alphabetically sort the different column names in the result. We can see in the result that the billing country for January invoices is NaN. However, there are values for the February invoices.

## 9. Concatenate tables with different column names

If we only want the matching columns between tables, we set the join argument to "inner". Its default value is equal to "outer", which is why concat by default will include all of the columns. Additionally, the sort argument has no effect when join equals "inner". The order of the columns will be the same as the input tables. Now the bill country column is gone and we're left with only the columns the tables have in common.

## 10. Using append method

Now let's briefly talk about append. Append is a simplified concat method. It supports the ignore\_index and sort arguments. However, it does not support keys or join. Join is always set to outer.

## 11. Append these tables

Let's combine these tables with the append method.

## 12. Append the tables

Append is a DataFrame method therefore, we list the "inv\_jan" table first then call the method. We add the other tables as a list, and set the ignore\_index and sort arguments similar to the concat method. In our output, we see null values for the billing country, except for February. Additionally, the index is adjusted as expected.

## 13. Let's practice!

With that, let's get some practice in!

**Daily XP650**

**Exercise**

**Exercise**

**Concatenation basics**

You have been given a few tables of data with musical track info for different albums from the metal band, *Metallica*. The track info comes from their *Ride The Lightning*, *Master Of Puppets*, and *St. Anger* albums. Try various features of the .concat() method by concatenating the tables vertically together in different ways.

The tables tracks\_master, tracks\_ride, and tracks\_st have loaded for you.

**Instructions 1/3**

**30 XP**

* [1](javascript:void(0))
  + Concatenate tracks\_master, tracks\_ride, and tracks\_st, in that order, setting sort to True.

 [2](javascript:void(0))

* Concatenate tracks\_master, tracks\_ride, and tracks\_st, where the index goes from 0 to n-1.

 [3](javascript:void(0))

* Concatenate tracks\_master, tracks\_ride, and tracks\_st, showing only columns that are in all tables.
* # Concatenate the tracks
* tracks\_from\_albums = pd.concat(\_\_\_\_,
* sort=True)
* print(tracks\_from\_albums)

# Merge the non\_mus\_tck and top\_invoices tables on tid tracks\_invoices = non\_mus\_tcks.merge(top\_invoices, on='tid', how='inner') print(tracks\_invoices) # Use .isin() to subset non\_mus\_tcks to rows with tid in tracks\_invoices top\_tracks = non\_mus\_tcks[non\_mus\_tcks['tid'].isin(tracks\_invoices['tid'])] print(top\_tracks) # Group the top\_tracks by gid and count the tid rows cnt\_by\_gid = top\_tracks.groupby(['gid'], as\_index=False).agg({'tid': 'count'}) # Merge the genres table to cnt\_by\_gid on gid and print print(cnt\_by\_gid.merge(genres, on='gid'))

# Concatenate the tracks

tracks\_from\_albums = pd.concat([tracks\_master, tracks\_ride, tracks\_st],

                               sort=True)

print(tracks\_from\_albums)

# Concatenate the tracks

tracks\_from\_albums = pd.concat([tracks\_master, tracks\_ride, tracks\_st],

sort=True)

print(tracks\_from\_albums)

aid composer gid mtid name tid u\_price

0 152 J.Hetfield/L.Ulrich 3 1 Battery 1853 0.99

1 152 K.Hammett 3 1 Master Of Puppets 1854 0.99

4 152 J.Hetfield/L.Ulrich 3 1 Disposable Heroes 1857 0.99

0 154 NaN 3 1 Fight Fire With Fire 1874 0.99

1 154 NaN 3 1 Ride The Lightning 1875 0.99

2 154 NaN 3 1 For Whom The Bell Tolls 1876 0.99

3 154 NaN 3 1 Fade To Black 1877 0.99

4 154 NaN 3 1 Trapped Under Ice 1878 0.99

0 155 NaN 3 1 Frantic 1882 0.99

1 155 NaN 3 1 St. Anger 1883 0.99

2 155 NaN 3 1 Some Kind Of Monster 1884 0.99

3 155 NaN 3 1 Dirty Window 1885 0.99

4 155 NaN 3 1 Invisible Kid 1886 0.99

# Concatenate the tracks so the index goes from 0 to n-1

tracks\_from\_albums = pd.concat([tracks\_master, tracks\_ride, tracks\_st],

                               ignore\_index=True,

                               sort=True)

print(tracks\_from\_albums)

# Merge the genres table to cnt\_by\_gid on gid and print

print(cnt\_by\_gid.merge(genres, on='gid'))

# Concatenate the tracks

tracks\_from\_albums = pd.concat([tracks\_master, tracks\_ride, tracks\_st],

sort=True)

print(tracks\_from\_albums)

aid composer gid mtid name tid u\_price

0 152 J.Hetfield/L.Ulrich 3 1 Battery 1853 0.99

1 152 K.Hammett 3 1 Master Of Puppets 1854 0.99

4 152 J.Hetfield/L.Ulrich 3 1 Disposable Heroes 1857 0.99

0 154 NaN 3 1 Fight Fire With Fire 1874 0.99

1 154 NaN 3 1 Ride The Lightning 1875 0.99

2 154 NaN 3 1 For Whom The Bell Tolls 1876 0.99

3 154 NaN 3 1 Fade To Black 1877 0.99

4 154 NaN 3 1 Trapped Under Ice 1878 0.99

0 155 NaN 3 1 Frantic 1882 0.99

1 155 NaN 3 1 St. Anger 1883 0.99

2 155 NaN 3 1 Some Kind Of Monster 1884 0.99

3 155 NaN 3 1 Dirty Window 1885 0.99

4 155 NaN 3 1 Invisible Kid 1886 0.99

<script.py> output:

aid composer gid mtid name tid u\_price

0 152 J.Hetfield/L.Ulrich 3 1 Battery 1853 0.99

1 152 K.Hammett 3 1 Master Of Puppets 1854 0.99

4 152 J.Hetfield/L.Ulrich 3 1 Disposable Heroes 1857 0.99

0 154 NaN 3 1 Fight Fire With Fire 1874 0.99

1 154 NaN 3 1 Ride The Lightning 1875 0.99

2 154 NaN 3 1 For Whom The Bell Tolls 1876 0.99

3 154 NaN 3 1 Fade To Black 1877 0.99

4 154 NaN 3 1 Trapped Under Ice 1878 0.99

0 155 NaN 3 1 Frantic 1882 0.99

1 155 NaN 3 1 St. Anger 1883 0.99

2 155 NaN 3 1 Some Kind Of Monster 1884 0.99

3 155 NaN 3 1 Dirty Window 1885 0.99

4 155 NaN 3 1 Invisible Kid 1886 0.99

# Concatenate the tracks so the index goes from 0 to n-1

tracks\_from\_albums = pd.concat([tracks\_master, tracks\_ride, tracks\_st],

ignore\_index=True,

sort=True)

print(tracks\_from\_albums)

aid composer gid mtid name tid u\_price

0 152 J.Hetfield/L.Ulrich 3 1 Battery 1853 0.99

1 152 K.Hammett 3 1 Master Of Puppets 1854 0.99

2 152 J.Hetfield/L.Ulrich 3 1 Disposable Heroes 1857 0.99

3 154 NaN 3 1 Fight Fire With Fire 1874 0.99

4 154 NaN 3 1 Ride The Lightning 1875 0.99

5 154 NaN 3 1 For Whom The Bell Tolls 1876 0.99

6 154 NaN 3 1 Fade To Black 1877 0.99

7 154 NaN 3 1 Trapped Under Ice 1878 0.99

8 155 NaN 3 1 Frantic 1882 0.99

9 155 NaN 3 1 St. Anger 1883 0.99

10 155 NaN 3 1 Some Kind Of Monster 1884 0.99

11 155 NaN 3 1 Dirty Window 1885 0.99

12 155 NaN 3 1 Invisible Kid 1886 0.99

# Concatenate the tracks, show only columns names that are in all tables

tracks\_from\_albums = pd.concat([tracks\_master, tracks\_ride, tracks\_st],

                               join= 'inner',

                               sort=True)

print(tracks\_from\_albums)

# Merge the non\_mus\_tck and top\_invoices tables on tid

tracks\_invoices = non\_mus\_tcks.merge(top\_invoices, on='tid', how='inner')

print(tracks\_invoices)

# Use .isin() to subset non\_mus\_tcks to rows with tid in tracks\_invoices

top\_tracks = non\_mus\_tcks[non\_mus\_tcks['tid'].isin(tracks\_invoices['tid'])]

print(top\_tracks)

# Group the top\_tracks by gid and count the tid rows

cnt\_by\_gid = top\_tracks.groupby(['gid'], as\_index=False).agg({'tid': 'count'})

# Merge the genres table to cnt\_by\_gid on gid and print

print(cnt\_by\_gid.merge(genres, on='gid'))

# Concatenate the tracks

tracks\_from\_albums = pd.concat([tracks\_master, tracks\_ride, tracks\_st],

sort=True)

print(tracks\_from\_albums)

aid composer gid mtid name tid u\_price

0 152 J.Hetfield/L.Ulrich 3 1 Battery 1853 0.99

1 152 K.Hammett 3 1 Master Of Puppets 1854 0.99

4 152 J.Hetfield/L.Ulrich 3 1 Disposable Heroes 1857 0.99

0 154 NaN 3 1 Fight Fire With Fire 1874 0.99

1 154 NaN 3 1 Ride The Lightning 1875 0.99

2 154 NaN 3 1 For Whom The Bell Tolls 1876 0.99

3 154 NaN 3 1 Fade To Black 1877 0.99

4 154 NaN 3 1 Trapped Under Ice 1878 0.99

0 155 NaN 3 1 Frantic 1882 0.99

1 155 NaN 3 1 St. Anger 1883 0.99

2 155 NaN 3 1 Some Kind Of Monster 1884 0.99

3 155 NaN 3 1 Dirty Window 1885 0.99

4 155 NaN 3 1 Invisible Kid 1886 0.99

<script.py> output:

aid composer gid mtid name tid u\_price

0 152 J.Hetfield/L.Ulrich 3 1 Battery 1853 0.99

1 152 K.Hammett 3 1 Master Of Puppets 1854 0.99

4 152 J.Hetfield/L.Ulrich 3 1 Disposable Heroes 1857 0.99

0 154 NaN 3 1 Fight Fire With Fire 1874 0.99

1 154 NaN 3 1 Ride The Lightning 1875 0.99

2 154 NaN 3 1 For Whom The Bell Tolls 1876 0.99

3 154 NaN 3 1 Fade To Black 1877 0.99

4 154 NaN 3 1 Trapped Under Ice 1878 0.99

0 155 NaN 3 1 Frantic 1882 0.99

1 155 NaN 3 1 St. Anger 1883 0.99

2 155 NaN 3 1 Some Kind Of Monster 1884 0.99

3 155 NaN 3 1 Dirty Window 1885 0.99

4 155 NaN 3 1 Invisible Kid 1886 0.99

# Concatenate the tracks so the index goes from 0 to n-1

tracks\_from\_albums = pd.concat([tracks\_master, tracks\_ride, tracks\_st],

ignore\_index=True,

sort=True)

print(tracks\_from\_albums)

aid composer gid mtid name tid u\_price

0 152 J.Hetfield/L.Ulrich 3 1 Battery 1853 0.99

1 152 K.Hammett 3 1 Master Of Puppets 1854 0.99

2 152 J.Hetfield/L.Ulrich 3 1 Disposable Heroes 1857 0.99

3 154 NaN 3 1 Fight Fire With Fire 1874 0.99

4 154 NaN 3 1 Ride The Lightning 1875 0.99

5 154 NaN 3 1 For Whom The Bell Tolls 1876 0.99

6 154 NaN 3 1 Fade To Black 1877 0.99

7 154 NaN 3 1 Trapped Under Ice 1878 0.99

8 155 NaN 3 1 Frantic 1882 0.99

9 155 NaN 3 1 St. Anger 1883 0.99

10 155 NaN 3 1 Some Kind Of Monster 1884 0.99

11 155 NaN 3 1 Dirty Window 1885 0.99

12 155 NaN 3 1 Invisible Kid 1886 0.99

<script.py> output:

aid composer gid mtid name tid u\_price

0 152 J.Hetfield/L.Ulrich 3 1 Battery 1853 0.99

1 152 K.Hammett 3 1 Master Of Puppets 1854 0.99

2 152 J.Hetfield/L.Ulrich 3 1 Disposable Heroes 1857 0.99

3 154 NaN 3 1 Fight Fire With Fire 1874 0.99

4 154 NaN 3 1 Ride The Lightning 1875 0.99

5 154 NaN 3 1 For Whom The Bell Tolls 1876 0.99

6 154 NaN 3 1 Fade To Black 1877 0.99

7 154 NaN 3 1 Trapped Under Ice 1878 0.99

8 155 NaN 3 1 Frantic 1882 0.99

9 155 NaN 3 1 St. Anger 1883 0.99

10 155 NaN 3 1 Some Kind Of Monster 1884 0.99

11 155 NaN 3 1 Dirty Window 1885 0.99

12 155 NaN 3 1 Invisible Kid 1886 0.99

# Concatenate the tracks, show only columns names that are in all tables

tracks\_from\_albums = pd.concat([tracks\_master, tracks\_ride, tracks\_st],

join= 'inner',

sort=True)

print(tracks\_from\_albums)

aid gid mtid name tid u\_price

0 152 3 1 Battery 1853 0.99

1 152 3 1 Master Of Puppets 1854 0.99

4 152 3 1 Disposable Heroes 1857 0.99

0 154 3 1 Fight Fire With Fire 1874 0.99

1 154 3 1 Ride The Lightning 1875 0.99

2 154 3 1 For Whom The Bell Tolls 1876 0.99

3 154 3 1 Fade To Black 1877 0.99

4 154 3 1 Trapped Under Ice 1878 0.99

0 155 3 1 Frantic 1882 0.99

1 155 3 1 St. Anger 1883 0.99

2 155 3 1 Some Kind Of Monster 1884 0.99

3 155 3 1 Dirty Window 1885 0.99

4 155 3 1 Invisible Kid 1886 0.99

<script.py> output:

aid gid mtid name tid u\_price

0 152 3 1 Battery 1853 0.99

1 152 3 1 Master Of Puppets 1854 0.99

4 152 3 1 Disposable Heroes 1857 0.99

0 154 3 1 Fight Fire With Fire 1874 0.99

1 154 3 1 Ride The Lightning 1875 0.99

2 154 3 1 For Whom The Bell Tolls 1876 0.99

3 154 3 1 Fade To Black 1877 0.99

4 154 3 1 Trapped Under Ice 1878 0.99

0 155 3 1 Frantic 1882 0.99

1 155 3 1 St. Anger 1883 0.99

2 155 3 1 Some Kind Of Monster 1884 0.99

3 155 3 1 Dirty Window 1885 0.99

4 155 3 1 Invisible Kid 1886 0.99

Great job! You've concatenated your first set of tables, adjusted the index, and altered the columns shown in the output. The .concat() method is a very flexible tool that is useful for combining data into a new dataset.

**Daily XP750**

**Exercise**

**Exercise**

**Concatenating with keys**

The leadership of the music streaming company has come to you and asked you for assistance in analyzing sales for a recent business quarter. They would like to know which month in the quarter saw the highest average invoice total. You have been given three tables with invoice data named inv\_jul, inv\_aug, and inv\_sep. Concatenate these tables into one to create a graph of the average monthly invoice total.

**Instructions**

**100 XP**

* Concatenate the three tables together vertically in order with the oldest month first, adding '7Jul', '8Aug', and '9Sep' as keys for their respective months, and save to variable avg\_inv\_by\_month.
* Use the .agg() method to find the average of the total column from the grouped invoices.
* Create a bar chart of avg\_inv\_by\_month.

# Concatenate the tables and add keys

inv\_jul\_thr\_sep = pd.concat(\_\_\_\_,

                            keys=\_\_\_\_)

# Group the invoices by the index keys and find avg of the total column

avg\_inv\_by\_month = inv\_jul\_thr\_sep.groupby(level=0).agg(\_\_\_\_)

# Bar plot of avg\_inv\_by\_month

avg\_inv\_by\_month.\_\_\_\_

plt.show()

# Concatenate the tracks, show only columns names that are in all tables tracks\_from\_albums = pd.concat([tracks\_master, tracks\_ride, tracks\_st], join= 'inner', sort=True) print(tracks\_from\_albums)

# Concatenate the tables and add keys

inv\_jul\_thr\_sep = pd.concat([inv\_jul, inv\_aug, inv\_sep], ignore\_index=False,

                            keys=['7Jul', '8Aug', '9Sep'])

# Group the invoices by the index keys and find avg of the total column

avg\_inv\_by\_month = inv\_jul\_thr\_sep.groupby(level=0).agg({'total', 'mean'})

# Bar plot of avg\_inv\_by\_month

avg\_inv\_by\_month.plot(kind='bar')

plt.show()

# Concatenate the tables and add keys

inv\_jul\_thr\_sep = pd.concat([inv\_jul, inv\_aug, inv\_sep], ignore\_index=False,

                            keys=['7Jul', '8Aug', '9Sep'])

# Group the invoices by the index keys and find avg of the total column

avg\_inv\_by\_month = inv\_jul\_thr\_sep.groupby(level=0).agg({'total': 'mean'})

print(avg\_inv\_by\_month)

# Bar plot of avg\_inv\_by\_month

avg\_inv\_by\_month.plot(kind='bar')

plt.show()

In [1]:

print(inv\_jul)

iid cid invoice\_date total bill\_ctry

0 42 51 2009-07-06 1.98 Sweden

1 43 53 2009-07-06 1.98 UK

2 44 55 2009-07-07 3.96 Australia

3 45 59 2009-07-08 5.94 India

4 46 6 2009-07-11 8.91 Czech Republic

5 47 15 2009-07-16 13.86 Canada

6 48 29 2009-07-24 0.99 Canada

7 126 35 2010-07-13 1.98 Portugal

8 127 37 2010-07-13 1.98 Germany

9 128 39 2010-07-14 3.96 France

10 129 43 2010-07-15 5.94 France

11 130 49 2010-07-18 8.91 Poland

12 131 58 2010-07-23 13.86 India

13 132 13 2010-07-31 0.99 Brazil

14 209 18 2011-07-07 0.99 USA

15 210 19 2011-07-20 1.98 USA

16 211 21 2011-07-20 1.98 USA

17 212 23 2011-07-21 3.96 USA

18 213 27 2011-07-22 5.94 USA

19 214 33 2011-07-25 8.91 Canada

20 215 42 2011-07-30 13.86 France

21 292 47 2012-07-05 13.86 Italy

22 293 2 2012-07-13 0.99 Germany

23 294 3 2012-07-26 1.98 Canada

24 295 5 2012-07-26 1.98 Czech Republic

25 296 7 2012-07-27 3.96 Austria

26 297 11 2012-07-28 5.94 Brazil

27 298 17 2012-07-31 10.91 USA

28 371 8 2013-07-02 1.98 Belgium

29 372 10 2013-07-02 1.98 Brazil

30 373 12 2013-07-03 3.96 Brazil

31 374 16 2013-07-04 5.94 USA

32 375 22 2013-07-07 8.91 USA

33 376 31 2013-07-12 13.86 Canada

34 377 45 2013-07-20 0.99 Hungary

# Concatenate the tables and add keys

inv\_jul\_thr\_sep = pd.concat([inv\_jul, inv\_aug, inv\_sep], ignore\_index=False,

keys=['7Jul', '8Aug', '9Sep'])

# Group the invoices by the index keys and find avg of the total column

avg\_inv\_by\_month = inv\_jul\_thr\_sep.groupby(level=0).agg({'total': 'mean'})

# Bar plot of avg\_inv\_by\_month

avg\_inv\_by\_month.plot(kind='bar')

plt.show()

# Concatenate the tables and add keys

inv\_jul\_thr\_sep = pd.concat([inv\_jul, inv\_aug, inv\_sep], ignore\_index=False,

keys=['7Jul', '8Aug', '9Sep'])

# Group the invoices by the index keys and find avg of the total column

avg\_inv\_by\_month = inv\_jul\_thr\_sep.groupby(level=0).agg({'total': 'mean'})

print(avg\_inv\_by\_month)

# Bar plot of avg\_inv\_by\_month

avg\_inv\_by\_month.plot(kind='bar')

plt.show()

total

7Jul 5.431

8Aug 5.660

9Sep 5.945

Way to come through! There are many ways to write code for this task. However, concatenating the tables with a key provides a hierarchical index that can be used for grouping. Once grouped, you can average the groups and create plots. You were able to find out that September had the highest average invoice total.

**Daily XP850**

**Exercise**

**Exercise**

**Using the append method**

The .concat() method is excellent when you need a lot of control over how concatenation is performed. However, if you do not need as much control, then the .append() method is another option. You'll try this method out by appending the track lists together from different *Metallica* albums. From there, you will merge it with the invoice\_items table to determine which track sold the most.

The tables tracks\_master, tracks\_ride, tracks\_st, and invoice\_items have loaded for you.

**Instructions**

**100 XP**

* Use the .append() method to combine (**in this order**) tracks\_ride, tracks\_master, and tracks\_st together vertically, and save to metallica\_tracks.
* Merge metallica\_tracks and invoice\_items on tid with an inner join, and save to tracks\_invoices.
* For each tid and name in tracks\_invoices, sum the quantity sold column, and save as tracks\_sold.
* Sort tracks\_sold in descending order by the quantity column, and print the table.
* # Use the .append() method to combine the tracks tables
* metallica\_tracks = \_\_\_\_.append(\_\_\_\_, sort=False)
* # Merge metallica\_tracks and invoice\_items
* tracks\_invoices = \_\_\_\_
* # For each tid and name sum the quantity sold
* tracks\_sold = tracks\_invoices.groupby(['tid','name']).agg(\_\_\_\_)
* # Sort in decending order by quantity and print the results
* print(tracks\_sold.sort\_values(\_\_\_\_))

# Concatenate the tables and add keys inv\_jul\_thr\_sep = pd.concat([inv\_jul, inv\_aug, inv\_sep], ignore\_index=False, keys=['7Jul', '8Aug', '9Sep']) # Group the invoices by the index keys and find avg of the total column avg\_inv\_by\_month = inv\_jul\_thr\_sep.groupby(level=0).agg({'total': 'mean'}) print(avg\_inv\_by\_month) # Bar plot of avg\_inv\_by\_month avg\_inv\_by\_month.plot(kind='bar') plt.show()

# Merge metallica\_tracks and invoice\_items

tracks\_invoices = metallica\_tracks.merge(invoice\_items, on='tid', how='inner')

# For each tid and name sum the quantity sold

tracks\_sold = tracks\_invoices.groupby(['tid','name']).agg({'quantity':'sum'})

# Sort in decending order by quantity and print the results

print(tracks\_sold.sort\_values('quantity', ascending=False))

# Use the .append() method to combine the tracks tables

metallica\_tracks = tracks\_master.append([tracks\_master, tracks\_st], sort=False)

# Merge metallica\_tracks and invoice\_items

tracks\_invoices = metallica\_tracks.merge(invoice\_items, on='tid', how='inner')

# For each tid and name sum the quantity sold

tracks\_sold = tracks\_invoices.groupby(['tid','name']).agg({'quantity':'sum'})

# Sort in decending order by quantity and print the results

print(tracks\_sold.sort\_values('quantity', ascending=False))

quantity

tid name

1853 Battery 4

1854 Master Of Puppets 2

1857 Disposable Heroes 2

1882 Frantic 1

1884 Some Kind Of Monster 1

1886 Invisible Kid 1

 Use the .append() method to combine the tracks tables

metallica\_tracks = tracks\_ride.append([tracks\_master, tracks\_st], sort=False)

# Merge metallica\_tracks and invoice\_items

tracks\_invoices = metallica\_tracks.merge(invoice\_items, on='tid', how='inner')

# For each tid and name sum the quantity sold

tracks\_sold = tracks\_invoices.groupby(['tid','name']).agg({'quantity':'sum'})

# Sort in decending order by quantity and print the results

print(tracks\_sold.sort\_values('quantity', ascending=False))

ror: name 'tid' is not defined

# Use the .append() method to combine the tracks tables

metallica\_tracks = tracks\_master.append([tracks\_master, tracks\_st], sort=False)

# Merge metallica\_tracks and invoice\_items

tracks\_invoices = metallica\_tracks.merge(invoice\_items, on='tid', how='inner')

# For each tid and name sum the quantity sold

tracks\_sold = tracks\_invoices.groupby(['tid','name']).agg({'quantity':'sum'})

# Sort in decending order by quantity and print the results

print(tracks\_sold.sort\_values('quantity', ascending=False))

quantity

tid name

1853 Battery 4

1854 Master Of Puppets 2

1857 Disposable Heroes 2

1882 Frantic 1

1884 Some Kind Of Monster 1

1886 Invisible Kid 1

# Use the .append() method to combine the tracks tables

metallica\_tracks = tracks\_ride.append([tracks\_master, tracks\_st], sort=False)

# Merge metallica\_tracks and invoice\_items

tracks\_invoices = metallica\_tracks.merge(invoice\_items, on='tid', how='inner')

# For each tid and name sum the quantity sold

tracks\_sold = tracks\_invoices.groupby(['tid','name']).agg({'quantity':'sum'})

# Sort in decending order by quantity and print the results

print(tracks\_sold.sort\_values('quantity', ascending=False))

quantity

tid name

1853 Battery 2

1876 For Whom The Bell Tolls 2

1854 Master Of Puppets 1

1857 Disposable Heroes 1

1875 Ride The Lightning 1

1877 Fade To Black 1

1882 Frantic 1

1884 Some Kind Of Monster 1

1886 Invisible Kid 1

<script.py> output:

quantity

tid name

1853 Battery 2

1876 For Whom The Bell Tolls 2

1854 Master Of Puppets 1

1857 Disposable Heroes 1

1875 Ride The Lightning 1

1877 Fade To Black 1

1882 Frantic 1

1884 Some Kind Of Monster 1

1886 Invisible Kid 1

Great work! Even though .append() is less flexible, it's also simpler than .concat(). It looks like \_Battery\_, and \_For Whom The Bell Tolls\_ were the most sold tracks.

**Daily XP950**

# Verifying integrity

**50 XP**

## 1. Verifying integrity

Welcome back. In this lesson, let's talk about verifying the integrity of our data.

## 2. Let's check our data

Both the merge and concat methods have special features that allow us to verify the structure of our data. When merging two tables, we might expect the tables to have a one-to-one relationship. However, one of the columns we are merging on may have a duplicated value, which will turn the relationship into a one-to-many. When concatenating tables vertically, we might unintentionally create duplicate records if a record exists in both tables. The validate and verify\_integrity arguments of the merge and concat methods respectively will allow us to verify the data.

## 3. Validating merges

Let's start with the merge method. If we provide the validate argument one of these key strings, it will validate the relationship between the two tables. For example, if we specify we want a one-to-one relationship, but it turns out the relationship is not one-to-one, then an error is raised. Let's try it out.

## 4. Merge dataset for example

In this example, we want to merge these two tables on the column "tid". Again, our data is from our music service. The first table is named "tracks", and the second is called "specs" for the technical specifications of each track. Each track should have one set of specifications, so this should be a one-to-one merge. However, notice that the specs table has two rows with a "tid" value equal to two. Therefore, merging these tables now becomes, unintentionally, a one-to-many relationship.

## 5. Merge validate: one\_to\_one

Let's merge the two tables with the tracks table on the left and specs on the right. Additionally, let's set the validate argument equal to one\_to\_one. In the result, a MergeError is raised. Python then tells us that the right table has duplicates, so it is not a one-to-one merge. We know that we should handle those duplicates properly before merging.

## 6. Merge validate: one\_to\_many

Now we'll merge album information with the tracks table. For every album there are multiple tracks, so this should be a one-to-many relationship. When we set the validate argument to "one\_to\_many" no error is raised.

## 7. Verifying concatenations

Let's now talk about the concat method. It has the argument verify\_integrity, which by default is False. However, if set to True, it will check if there are duplicate values in the index and raise an error if there are. It will only check the index values and not the columns.

## 8. Dataset for .concat() example

To try out this feature, we will attempt to concatenate these two tables. They are the February and March invoice data shown in a previous video. However, both tables were modified so the index contains invoice IDs. Notice that invoice ID number 9 is in both tables.

## 9. Verifying concatenation: example

Let's try to concatenate the two tables together with the verify\_integrity argument set to True. The concat method raises a ValueError stating that the indexes have overlapping values. Now let's try to concatenate the two tables again with the verify\_integrity set back to the default value of False. The concat method now returns a combined table with the invoice ID of number 9 repeated twice.

## 10. Why verify integrity and what to do

Often our data is not clean, and it may not always be evident if data has the expected structure. Therefore, verifying this structure is useful, saving us from having a mean skewed by duplicate values, or from creating inaccurate plots. If you receive a MergeError or a ValueError, you can fix the incorrect data or drop duplicate rows. In general, you should look to correct the issue.

## 11. Let's practice!

Time for some practice!

**Daily XP1000**

**Exercise**

**Exercise**

**Validating a merge**

You have been given 2 tables, artists, and albums. Use the IPython shell to merge them using artists.merge(albums, on='artid').head(). Adjust the validate argument to answer which statement is ***False***.

**Instructions**

**50 XP**

**Possible Answers**In [7]:

artists.merge(albums,on='artid',validate='one\_to\_many').head()

Out[7]:

artid name aid title

0 1 AC/DC 1 For Those About To Rock We Salute You

1 1 AC/DC 4 Let There Be Rock

2 2 Accept 2 Balls to the Wall

3 2 Accept 3 Restless and Wild

4 3 Aerosmith 5 Big Ones

* 

You can use 'many\_to\_many' without an error, since there is a duplicate key in one of the tables.

* 

You can use 'one\_to\_many' without error, since there is a duplicate key in the right table.

* 

You can use 'many\_to\_one' without an error, since there is a duplicate key in the left table.

This is the answer with Merge Errors : merge keys not unique in right dataset; not a many to one merge

# Use the .append() method to combine the tracks tables metallica\_tracks = tracks\_ride.append([tracks\_master, tracks\_st], sort=False) # Merge metallica\_tracks and invoice\_items tracks\_invoices = metallica\_tracks.merge(invoice\_items, on='tid', how='inner') # For each tid and name sum the quantity sold tracks\_sold = tracks\_invoices.groupby(['tid','name']).agg({'quantity':'sum'}) # Sort in decending order by quantity and print the results print(tracks\_sold.sort\_values('quantity', ascending=False))

In [8]:

artists.merge(albums,on='artid',validate='many\_to\_one').head()

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 1, in <module>

artists.merge(albums,on='artid',validate='many\_to\_one').head()

File "<stdin>", line 9190, in merge

return merge(

File "<stdin>", line 106, in merge

op = \_MergeOperation(

File "<stdin>", line 709, in \_\_init\_\_

self.\_validate(validate)

File "<stdin>", line 1441, in \_validate

raise MergeError(

pandas.errors.MergeError: Merge keys are not unique in right dataset; not a many-to-one merge

In [9]:

artists.merge(albums,on='artid',validate='many\_to\_many').head()

Out[9]:

artid name aid title

0 1 AC/DC 1 For Those About To Rock We Salute You

1 1 AC/DC 4 Let There Be Rock

2 2 Accept 2 Balls to the Wall

3 2 Accept 3 Restless and Wild

4 3 Aerosmith 5 Big Ones

That's correct! This statement is false. There is a duplicate value in the artid column in the albums table, which is the right table in this merge. Therefore, setting validate equal to 'many\_to\_one' or 'one\_to\_one' will raise an error, making this statement false.

**Daily XP1050**

**Exercise**

**Exercise**

**Concatenate and merge to find common songs**

The senior leadership of the streaming service is requesting your help again. You are given the historical files for a popular playlist in the classical music genre in 2018 and 2019. Additionally, you are given a similar set of files for the most popular pop music genre playlist on the streaming service in 2018 and 2019. Your goal is to concatenate the respective files to make a large classical playlist table and overall popular music table. Then filter the classical music table using a semi join to return only the most popular classical music tracks.

The tables classic\_18, classic\_19, and pop\_18, pop\_19 have been loaded for you. Additionally, pandas has been loaded as pd.

**Instructions 1/2**

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* Concatenate the classic\_18 and classic\_19 tables vertically where the index goes from 0 to n-1, and save to classic\_18\_19.
* Concatenate the pop\_18 and pop\_19 tables vertically where the index goes from 0 to n-1, and save to pop\_18\_19.

# Concatenate the classic tables vertically

classic\_18\_19 = \_\_\_\_

# Concatenate the pop tables vertically

pop\_18\_19 = \_\_\_\_

# Use the .append() method to combine the tracks tables metallica\_tracks = tracks\_ride.append([tracks\_master, tracks\_st], sort=False) # Merge metallica\_tracks and invoice\_items tracks\_invoices = metallica\_tracks.merge(invoice\_items, on='tid', how='inner') # For each tid and name sum the quantity sold tracks\_sold = tracks\_invoices.groupby(['tid','name']).agg({'quantity':'sum'}) # Sort in decending order by quantity and print the results print(tracks\_sold.sort\_values('quantity', ascending=False))

# Concatenate the classic tables vertically

classic\_18\_19 = pd.concat([classic\_18, classic\_19], ignore\_index=True)

print(classic\_18\_19)

# Concatenate the pop tables vertically

pop\_18\_19 = pd.concat([pop\_18, pop\_19], ignore\_index=True)

print(pop\_18\_19)

# Concatenate the classic tables vertically

classic\_18\_19 = pd.concat([classic\_18, classic\_19], ignore\_index=True)

print(classic\_18\_19)

# Concatenate the pop tables vertically

pop\_18\_19 = pd.concat([pop\_18, pop\_19], ignore\_index=True)

print(pop\_18\_19)

pid tid

0 12 3483

1 12 3416

2 12 3489

3 12 3479

4 12 3440

5 12 3414

6 12 3433

7 12 3491

8 12 3422

9 12 3417

10 12 3439

11 12 3447

12 12 3454

13 12 3497

14 12 3488

15 12 3501

16 12 3484

17 12 3438

18 12 3409

19 12 3419

20 12 3444

21 12 3445

22 12 3495

23 12 3449

24 12 3450

25 12 3482

26 12 3434

27 12 3448

28 12 3499

29 12 3425

30 12 3493

31 12 3446

32 12 3432

33 12 3410

34 12 3441

35 12 3407

36 12 3436

37 12 3418

38 12 3415

39 12 3452

40 12 3403

41 12 3405

42 12 3413

43 12 3424

44 12 3443

45 12 3430

46 12 3487

47 12 3494

48 12 3437

49 12 3420

50 12 3435

51 12 3502

pid tid

0 1 3063

1 1 2712

2 1 2641

3 1 2271

4 1 919

.. ... ...

371 1 2942

372 1 2463

373 1 2459

374 1 1540

375 1 2060

[376 rows x 2 columns]

**Daily XP1100**

**Exercise**

**Exercise**

**Concatenate and merge to find common songs**

The senior leadership of the streaming service is requesting your help again. You are given the historical files for a popular playlist in the classical music genre in 2018 and 2019. Additionally, you are given a similar set of files for the most popular pop music genre playlist on the streaming service in 2018 and 2019. Your goal is to concatenate the respective files to make a large classical playlist table and overall popular music table. Then filter the classical music table using a semi join to return only the most popular classical music tracks.

The tables classic\_18, classic\_19, and pop\_18, pop\_19 have been loaded for you. Additionally, pandas has been loaded as pd.

**Instructions 2/2**

**50 XP**

* [2](javascript:void(0))
* With classic\_18\_19 on the left, merge it with pop\_18\_19 on tid using an inner join.
* Use .isin() to filter classic\_18\_19 where tid is in classic\_pop.

# Concatenate the classic tables vertically

classic\_18\_19 = pd.concat([classic\_18, classic\_19], ignore\_index=True)

# Concatenate the pop tables vertically

pop\_18\_19 = pd.concat([pop\_18, pop\_19], ignore\_index=True)

# Merge classic\_18\_19 with pop\_18\_19

classic\_pop = \_\_\_\_

# Using .isin(), filter classic\_18\_19 rows where tid is in classic\_pop

popular\_classic = classic\_18\_19[classic\_18\_19[\_\_\_\_].isin(\_\_\_\_)]

# Print popular chart

print(popular\_classic)

* With classic\_18\_19 on the left, merge it with pop\_18\_19 on tid using an inner join.
* Use .isin() to filter classic\_18\_19 where tid is in classic\_pop.

# Concatenate the classic tables vertically

classic\_18\_19 = pd.concat([classic\_18, classic\_19], ignore\_index=True)

# Concatenate the pop tables vertically

pop\_18\_19 = pd.concat([pop\_18, pop\_19], ignore\_index=True)

# Merge classic\_18\_19 with pop\_18\_19

classic\_pop = classic\_18\_19.merge(pop\_18\_19, on='tid', how='inner')

# Using .isin(), filter classic\_18\_19 rows where tid is in classic\_pop

popular\_classic = classic\_18\_19[classic\_18\_19['tid'].isin(classic\_pop['tid'])]

# Print popular chart

print(popular\_classic)

# Concatenate the classic tables vertically

classic\_18\_19 = pd.concat([classic\_18, classic\_19], ignore\_index=True)

# Concatenate the pop tables vertically

pop\_18\_19 = pd.concat([pop\_18, pop\_19], ignore\_index=True)

# Merge classic\_18\_19 with pop\_18\_19

classic\_pop = classic\_18\_19.merge(pop\_18\_19, on='tid', how='inner')

# Using .isin(), filter classic\_18\_19 rows where tid is in classic\_pop

popular\_classic = classic\_18\_19[classic\_18\_19['tid'].isin(classic\_pop['tid'])]

# Print popular chart

print(popular\_classic)

pid tid

3 12 3479

10 12 3439

21 12 3445

23 12 3449

48 12 3437

50 12 3435

Excellent work! In this exercise, you demonstrated many of the concepts discussed in this chapter, including concatenation, and semi joins. You now have experience combining data vertically and using semi- and anti joins. Time to move on to the next chapter!

In this final chapter, you’ll step up a gear and learn to apply pandas' specialized methods for merging time-series and ordered data together with real-world financial and economic data from the city of Chicago. You’ll also learn how to query resulting tables using a SQL-style format, and unpivot data using the melt method.

**Daily XP1150**

# Using merge\_ordered()

**50 XP**

## 1. Using merge\_ordered()

Welcome back! In this last chapter we'll start discussing merge\_ordered(). This method can merge time-series and other ordered data.

## 2. merge\_ordered()

The merge\_ordered method will allow us to merge the left and right tables shown here. We can see the output of the merge when we merge on the "C" column. The results are similar to the standard merge method with an outer join, but here that the results are sorted. The sorted results make this a useful method for ordered or time-series data.

## 3. Method comparison

So, let's first give context to this method. It has many of the same arguments we have already covered with the merge method. They both contain arguments to allow us to merge two tables on different columns. Both methods support different types of joins. Although, the default for the merge method is "inner", it is "outer" for merge\_order method. Also, both methods support suffixes for overlapping column names. However, how you call each of the methods is different. Earlier in the course, we called the merge method by first listing a table and calling the method afterward. For merge\_ordered(), you'll need to first call pandas then merge\_ordered(). Let's look at an example.

## 4. Financial dataset

In this chapter, we will be working with financial, macroeconomic, and stock market data.

1. 1 Photo by Markus Spiske on Unsplash

## 5. Stock data

In this example, we have a table of the stock prices of the Apple corporation from February to June 2007. We also have a table of the stock price for McDonald's corporation from January to May 2007, and we want to merge them.

## 6. Merging stock data

The first two arguments are the left and right tables. We set the "on" argument equal to date. Finally, we set the suffixes argument to determine which table the data originated. This results in a table sorted by date. There isn't a value for Apple in January or a value for McDonald's for June since values for these time periods are not available in the two original tables.

## 7. Forward fill

We can fill in this missing data using a technique called forward filling. It will interpolate missing data by filling the missing values with the previous value. In the table shown here, the second and fourth rows of column B are filled with the values of B in the rows proceeding them.

## 8. Forward fill example

Going back to our stock example from before, we now set the fill\_method argument to "ffill" for forward fill. In the result, notice that the missing value for McDonald's in the last row is now filled in with the row before it. The table from before is shown on the right for easier comparison. Notice the missing value for Apple in the first row is still missing since there isn't a row before the first row to copy into the missing value for Apple.

## 9. When to use merge\_ordered()?

You might think about using the merge\_ordered method instead of the regular merge method when you are working with order or time-series data like in our example. Additionally, the fill forward feature is useful for handling missing data, as most machine learning algorithms require that there are no missing values.

## 10. Let's practice!

Time to practice using the method and add it to your toolbox!

**Daily XP50**

**Exercise**

**Exercise**

**Correlation between GDP and S&P500**

In this exercise, you want to analyze stock returns from the S&P 500. You believe there may be a relationship between the returns of the S&P 500 and the GDP of the US. Merge the different datasets together to compute the correlation.

Two tables have been provided for you, named sp500, and gdp. As always, pandas has been imported for you as pd.

**Instructions 1/3**

**35 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))
* Use merge\_ordered() to merge gdp and sp500 using a left join on year and date. Save the results as gdp\_sp500.
* Print gdp\_sp500 and look at the returns for the year 2018.
* # Use merge\_ordered() to merge gdp and sp500 on year and date
* gdp\_sp500 = pd.merge\_ordered(\_\_\_\_, \_\_\_\_, left\_on=\_\_\_\_, right\_on=\_\_\_\_,
* how=\_\_\_\_)
* # Print gdp\_sp500
* print(\_\_\_\_)

# Concatenate the classic tables vertically classic\_18\_19 = pd.concat([classic\_18, classic\_19], ignore\_index=True) # Concatenate the pop tables vertically pop\_18\_19 = pd.concat([pop\_18, pop\_19], ignore\_index=True) # Merge classic\_18\_19 with pop\_18\_19 classic\_pop = classic\_18\_19.merge(pop\_18\_19, on='tid', how='inner') # Using .isin(), filter classic\_18\_19 rows where tid is in classic\_pop popular\_classic = classic\_18\_19[classic\_18\_19['tid'].isin(classic\_pop['tid'])] # Print popular chart print(popular\_classic)

# Use merge\_ordered() to merge gdp and sp500 on year and date

gdp\_sp500 = pd.merge\_ordered(gdp, sp500, left\_on='year', right\_on='date',

                             how='left')

# Print gdp\_sp500

print(gdp\_sp500)

# Use merge\_ordered() to merge gdp and sp500 on year and date

gdp\_sp500 = pd.merge\_ordered(gdp, sp500, left\_on='year', right\_on='date',

how='left')

# Print gdp\_sp500

print(gdp\_sp500)

country code year gdp date returns

0 USA 2010 1.499e+13 2010.0 12.78

1 USA 2011 1.554e+13 2011.0 0.00

2 USA 2012 1.620e+13 2012.0 13.41

3 USA 2012 1.620e+13 2012.0 13.41

4 USA 2013 1.678e+13 2013.0 29.60

5 USA 2014 1.752e+13 2014.0 11.39

6 USA 2015 1.822e+13 2015.0 -0.73

7 USA 2016 1.871e+13 2016.0 9.54

8 USA 2017 1.949e+13 2017.0 19.42

9 USA 2018 2.049e+13 NaN NaN

Use merge\_ordered(), again similar to before, to merge gdp and sp500 use the function's ability to interpolate missing data to forward fill the missing value for returns, assigning this table to the variable gdp\_sp500.

# Use merge\_ordered() to merge gdp and sp500, interpolate missing value

gdp\_sp500 = pd.merge\_ordered(\_\_\_\_)

# Print gdp\_sp500

print (gdp\_sp500)

# Use merge\_ordered() to merge gdp and sp500, interpolate missing value

gdp\_sp500 = pd.merge\_ordered(gdp, sp500, left\_on='year', right\_on='date',

                             how='left', fill\_method= 'ffill')

# Print gdp\_sp500

print (gdp\_sp500)

# Use merge\_ordered() to merge gdp and sp500 on year and date

gdp\_sp500 = pd.merge\_ordered(gdp, sp500, left\_on='year', right\_on='date',

how='left')

# Print gdp\_sp500

print(gdp\_sp500)

country code year gdp date returns

0 USA 2010 1.499e+13 2010.0 12.78

1 USA 2011 1.554e+13 2011.0 0.00

2 USA 2012 1.620e+13 2012.0 13.41

3 USA 2012 1.620e+13 2012.0 13.41

4 USA 2013 1.678e+13 2013.0 29.60

5 USA 2014 1.752e+13 2014.0 11.39

6 USA 2015 1.822e+13 2015.0 -0.73

7 USA 2016 1.871e+13 2016.0 9.54

8 USA 2017 1.949e+13 2017.0 19.42

9 USA 2018 2.049e+13 NaN NaN

<script.py> output:

country code year gdp date returns

0 USA 2010 1.499e+13 2010.0 12.78

1 USA 2011 1.554e+13 2011.0 0.00

2 USA 2012 1.620e+13 2012.0 13.41

3 USA 2012 1.620e+13 2012.0 13.41

4 USA 2013 1.678e+13 2013.0 29.60

5 USA 2014 1.752e+13 2014.0 11.39

6 USA 2015 1.822e+13 2015.0 -0.73

7 USA 2016 1.871e+13 2016.0 9.54

8 USA 2017 1.949e+13 2017.0 19.42

9 USA 2018 2.049e+13 NaN NaN

# Use merge\_ordered() to merge gdp and sp500 on year and date

gdp\_sp500 = pd.merge\_ordered(gdp, sp500, left\_on='year', right\_on='date',

how='left')

# Print gdp\_sp500

print(gdp\_sp500)

country code year gdp date returns

0 USA 2010 1.499e+13 2010.0 12.78

1 USA 2011 1.554e+13 2011.0 0.00

2 USA 2012 1.620e+13 2012.0 13.41

3 USA 2012 1.620e+13 2012.0 13.41

4 USA 2013 1.678e+13 2013.0 29.60

5 USA 2014 1.752e+13 2014.0 11.39

6 USA 2015 1.822e+13 2015.0 -0.73

7 USA 2016 1.871e+13 2016.0 9.54

8 USA 2017 1.949e+13 2017.0 19.42

9 USA 2018 2.049e+13 NaN NaN

<script.py> output:

country code year gdp date returns

0 USA 2010 1.499e+13 2010.0 12.78

1 USA 2011 1.554e+13 2011.0 0.00

2 USA 2012 1.620e+13 2012.0 13.41

3 USA 2012 1.620e+13 2012.0 13.41

4 USA 2013 1.678e+13 2013.0 29.60

5 USA 2014 1.752e+13 2014.0 11.39

6 USA 2015 1.822e+13 2015.0 -0.73

7 USA 2016 1.871e+13 2016.0 9.54

8 USA 2017 1.949e+13 2017.0 19.42

9 USA 2018 2.049e+13 NaN NaN

# Use merge\_ordered() to merge gdp and sp500, interpolate missing value

gdp\_sp500 = pd.merge\_ordered(gdp, sp500, left\_on='year', right\_on='date',

how='left', fill\_method= 'ffill')

# Print gdp\_sp500

print (gdp\_sp500)

country code year gdp date returns

0 USA 2010 1.499e+13 2010 12.78

1 USA 2011 1.554e+13 2011 0.00

2 USA 2012 1.620e+13 2012 13.41

3 USA 2012 1.620e+13 2012 13.41

4 USA 2013 1.678e+13 2013 29.60

5 USA 2014 1.752e+13 2014 11.39

6 USA 2015 1.822e+13 2015 -0.73

7 USA 2016 1.871e+13 2016 9.54

8 USA 2017 1.949e+13 2017 19.42

9 USA 2018 2.049e+13 2017 19.42

# Use merge\_ordered() to merge gdp and sp500, interpolate missing value

gdp\_sp500 = pd.merge\_ordered(gdp, sp500, left\_on='year', right\_on='date',

                             how='left',  fill\_method='ffill')

# Subset the gdp and returns columns

gdp\_returns = gdp\_sp500[['gdp','returns']]

# Print gdp\_returns correlation

print(gdp\_returns)

# Use merge\_ordered() to merge gdp and sp500, interpolate missing value

gdp\_sp500 = pd.merge\_ordered(gdp, sp500, left\_on='year', right\_on='date',

how='left', fill\_method='ffill')

# Subset the gdp and returns columns

gdp\_returns = gdp\_sp500[['gdp','returns']]

# Print gdp\_returns correlation

print(gdp\_returns)

gdp returns

0 1.499e+13 12.78

1 1.554e+13 0.00

2 1.620e+13 13.41

3 1.620e+13 13.41

4 1.678e+13 29.60

5 1.752e+13 11.39

6 1.822e+13 -0.73

7 1.871e+13 9.54

8 1.949e+13 19.42

9 2.049e+13 19.42

# Use merge\_ordered() to merge gdp and sp500, interpolate missing value

gdp\_sp500 = pd.merge\_ordered(gdp, sp500, left\_on='year', right\_on='date',

how='left', fill\_method='ffill')

# Subset the gdp and returns columns

gdp\_returns = gdp\_sp500[['gdp','returns']]

# Print gdp\_returns correlation

print(gdp\_returns.corr())

gdp returns

gdp 1.000 0.212

returns 0.212 1.000

Awesome work! You can see the different aspects of merge\_ordered() and how you might use it on data that can be ordered. By using this function, you were able to fill in the missing data from 2019. Finally, the correlation of 0.21 between the GDP and S&P500 is low to moderate at best. You may want to find another predictor if you plan to play in the stock market.

**Daily XP150**

**Exercise**

**Exercise**

**Phillips curve using merge\_ordered()**

There is an economic theory developed by A. W. Phillips which states that inflation and unemployment have an inverse relationship. The theory claims that with economic growth comes inflation, which in turn should lead to more jobs and less unemployment.

You will take two tables of data from the U.S. Bureau of Labor Statistics, containing unemployment and inflation data over different periods, and create a Phillips curve. The tables have different frequencies. One table has a data entry every six months, while the other has a data entry every month. You will need to use the entries where you have data within both tables.

The tables unemployment and inflation have been loaded for you.

**Instructions**

**100 XP**

* Use merge\_ordered() to merge the inflation and unemployment tables on date with an inner join, and save the results as inflation\_unemploy.
* Print the inflation\_unemploy variable.
* Using inflation\_unemploy, create a scatter plot with unemployment\_rate on the horizontal axis and cpi (inflation) on the vertical axis.
* # Use merge\_ordered() to merge inflation, unemployment with inner join
* inflation\_unemploy = \_\_\_\_
* # Print inflation\_unemploy
* \_\_\_\_
* # Plot a scatter plot of unemployment\_rate vs cpi of inflation\_unemploy
* inflation\_unemploy.plot(\_\_\_\_)
* plt.show()

# Use merge\_ordered() to merge gdp and sp500, interpolate missing value gdp\_sp500 = pd.merge\_ordered(gdp, sp500, left\_on='year', right\_on='date', how='left', fill\_method='ffill') # Subset the gdp and returns columns gdp\_returns = gdp\_sp500[['gdp','returns']] # Print gdp\_returns correlation print(gdp\_returns.corr())

# Use merge\_ordered() to merge inflation, unemployment with inner join

inflation\_unemploy = pd.merge\_ordered(inflation, unemployment, on='date', how='inner')

# Print inflation\_unemploy

print(inflation\_unemploy)

# Plot a scatter plot of unemployment\_rate vs cpi of inflation\_unemploy

inflation\_unemploy.plot(x='unemployment\_rate', y='cpi', kind='scatter')

plt.show()

# Use merge\_ordered() to merge inflation, unemployment with inner join

inflation\_unemploy = pd.merge\_ordered(inflation, unemployment, on='date', how='inner')

# Print inflation\_unemploy

print(inflation\_unemploy)

# Plot a scatter plot of unemployment\_rate vs cpi of inflation\_unemploy

inflation\_unemploy.plot(x='unemployment\_rate', y='cpi', kind='scatter')

plt.show()

date cpi seriesid data\_type unemployment\_rate

0 2014-01-01 235.288 CUSR0000SA0 SEASONALLY ADJUSTED INDEX 6.7

1 2014-06-01 237.231 CUSR0000SA0 SEASONALLY ADJUSTED INDEX 6.1

2 2015-01-01 234.718 CUSR0000SA0 SEASONALLY ADJUSTED INDEX 5.6

3 2015-06-01 237.684 CUSR0000SA0 SEASONALLY ADJUSTED INDEX 5.3

4 2016-01-01 237.833 CUSR0000SA0 SEASONALLY ADJUSTED INDEX 5.0

5 2016-06-01 240.167 CUSR0000SA0 SEASONALLY ADJUSTED INDEX 4.9

6 2017-01-01 243.780 CUSR0000SA0 SEASONALLY ADJUSTED INDEX 4.7

7 2017-06-01 244.182 CUSR0000SA0 SEASONALLY ADJUSTED INDEX 4.3

8 2018-01-01 248.884 CUSR0000SA0 SEASONALLY ADJUSTED INDEX 4.1

9 2018-06-01 251.134 CUSR0000SA0 SEASONALLY ADJUSTED INDEX 4.0

Great work! You created a Phillips curve. There are critics of the curve, but what is more important in this example is that you were able to use entries where you had entries in both tables by using an inner join. You might ask why not use the default outer join and use forward fill to fill to estimate the missing variables. You might choose differently. In this case, instead of showing an estimated unemployment rate (which is a continually changing measure) for five periods, that data was dropped from the plot.

**Daily XP250**

**Exercise**

**Exercise**

**merge\_ordered() caution, multiple columns**

When using merge\_ordered() to merge on multiple columns, the order is important when you combine it with the forward fill feature. The function sorts the merge on columns in the order provided. In this exercise, we will merge GDP and population data from the World Bank for the Australia and Sweden, reversing the order of the merge on columns. The frequency of the series are different, the GDP values are quarterly, and the population is yearly. Use the forward fill feature to fill in the missing data. Depending on the order provided, the fill forward will use unintended data to fill in the missing values.

The tables gdp and pop have been loaded.

**Instructions 1/2**

**50 XP**

* [1](javascript:void(0))
  + Use merge\_ordered() on gdp and pop, merging on columns date and country with the fill feature, save to ctry\_date.

 [2](javascript:void(0))

* Perform the same merge of gdp and pop, but join on country and date (**reverse of step 1**) with the fill feature, saving this as date\_ctry.

# Merge gdp and pop on date and country with fill and notice rows 2 and 3

ctry\_date = pd.merge\_ordered(\_\_\_\_,

                             fill\_method='ffill')

# Print ctry\_date

print(ctry\_date)

# Use merge\_ordered() to merge inflation, unemployment with inner join inflation\_unemploy = pd.merge\_ordered(inflation, unemployment, on='date', how='inner') # Print inflation\_unemploy print(inflation\_unemploy) # Plot a scatter plot of unemployment\_rate vs cpi of inflation\_unemploy inflation\_unemploy.plot(x='unemployment\_rate', y='cpi', kind='scatter') plt.show()

# Merge gdp and pop on date and country with fill and notice rows 2 and 3

ctry\_date = pd.merge\_ordered(gdp, pop, on=['date','country'],

                             fill\_method='ffill')

# Print ctry\_date

print(ctry\_date)

# Merge gdp and pop on date and country with fill and notice rows 2 and 3

ctry\_date = pd.merge\_ordered(gdp, pop, on=['date','country'],

fill\_method='ffill')

# Print ctry\_date

print(ctry\_date)

date country gdp series\_code\_x pop series\_code\_y

0 1990-01-01 Australia 158051.132 NYGDPMKTPSAKD 17065100 SP.POP.TOTL

1 1990-01-01 Sweden 79837.846 NYGDPMKTPSAKD 8558835 SP.POP.TOTL

2 1990-04-01 Australia 158263.582 NYGDPMKTPSAKD 8558835 SP.POP.TOTL

3 1990-04-01 Sweden 80582.286 NYGDPMKTPSAKD 8558835 SP.POP.TOTL

4 1990-07-01 Australia 157329.279 NYGDPMKTPSAKD 8558835 SP.POP.TOTL

5 1990-07-01 Sweden 79974.360 NYGDPMKTPSAKD 8558835 SP.POP.TOTL

6 1990-09-01 Australia 158240.678 NYGDPMKTPSAKD 8558835 SP.POP.TOTL

7 1990-09-01 Sweden 80106.497 NYGDPMKTPSAKD 8558835 SP.POP.TOTL

8 1991-01-01 Australia 156195.954 NYGDPMKTPSAKD 17284000 SP.POP.TOTL

9 1991-01-01 Sweden 79524.242 NYGDPMKTPSAKD 8617375 SP.POP.TOTL

10 1991-04-01 Australia 155989.033 NYGDPMKTPSAKD 8617375 SP.POP.TOTL

11 1991-04-01 Sweden 79073.059 NYGDPMKTPSAKD 8617375 SP.POP.TOTL

12 1991-07-01 Australia 156635.858 NYGDPMKTPSAKD 8617375 SP.POP.TOTL

13 1991-07-01 Sweden 79084.770 NYGDPMKTPSAKD 8617375 SP.POP.TOTL

14 1991-09-01 Australia 156744.057 NYGDPMKTPSAKD 8617375 SP.POP.TOTL

15 1991-09-01 Sweden 79740.606 NYGDPMKTPSAKD 8617375 SP.POP.TOTL

16 1992-01-01 Australia 157916.081 NYGDPMKTPSAKD 17495000 SP.POP.TOTL

17 1992-01-01 Sweden 79390.922 NYGDPMKTPSAKD 8668067 SP.POP.TOTL

18 1992-04-01 Australia 159047.827 NYGDPMKTPSAKD 8668067 SP.POP.TOTL

19 1992-04-01 Sweden 79060.283 NYGDPMKTPSAKD 8668067 SP.POP.TOTL

20 1992-07-01 Australia 160658.176 NYGDPMKTPSAKD 8668067 SP.POP.TOTL

21 1992-07-01 Sweden 78904.605 NYGDPMKTPSAKD 8668067 SP.POP.TOTL

22 1992-09-01 Australia 163960.221 NYGDPMKTPSAKD 8668067 SP.POP.TOTL

23 1992-09-01 Sweden 76996.837 NYGDPMKTPSAKD 8668067 SP.POP.TOTL

24 1993-01-01 Australia 165097.495 NYGDPMKTPSAKD 17667000 SP.POP.TOTL

25 1993-01-01 Sweden 75783.588 NYGDPMKTPSAKD 8718561 SP.POP.TOTL

26 1993-04-01 Australia 166027.059 NYGDPMKTPSAKD 8718561 SP.POP.TOTL

27 1993-04-01 Sweden 76708.548 NYGDPMKTPSAKD 8718561 SP.POP.TOTL

28 1993-07-01 Australia 166203.179 NYGDPMKTPSAKD 8718561 SP.POP.TOTL

29 1993-07-01 Sweden 77662.018 NYGDPMKTPSAKD 8718561 SP.POP.TOTL

30 1993-09-01 Australia 169279.348 NYGDPMKTPSAKD 8718561 SP.POP.TOTL

31 1993-09-01 Sweden 77703.304 NYGDPMKTPSAKD 8718561 SP.POP.TOTL

date\_ctry = pd.merge\_ordered(gdp, pop, on=['country','date'],

                             fill\_method='ffill')

# Print date\_ctry

print(date\_ctry)

# Merge gdp and pop on country and date with fill

date\_ctry = pd.merge\_ordered(gdp, pop, on=['country','date'],

fill\_method='ffill')

# Print date\_ctry

print(date\_ctry)

date country gdp series\_code\_x pop series\_code\_y

0 1990-01-01 Australia 158051.132 NYGDPMKTPSAKD 17065100 SP.POP.TOTL

1 1990-04-01 Australia 158263.582 NYGDPMKTPSAKD 17065100 SP.POP.TOTL

2 1990-07-01 Australia 157329.279 NYGDPMKTPSAKD 17065100 SP.POP.TOTL

3 1990-09-01 Australia 158240.678 NYGDPMKTPSAKD 17065100 SP.POP.TOTL

4 1991-01-01 Australia 156195.954 NYGDPMKTPSAKD 17284000 SP.POP.TOTL

5 1991-04-01 Australia 155989.033 NYGDPMKTPSAKD 17284000 SP.POP.TOTL

6 1991-07-01 Australia 156635.858 NYGDPMKTPSAKD 17284000 SP.POP.TOTL

7 1991-09-01 Australia 156744.057 NYGDPMKTPSAKD 17284000 SP.POP.TOTL

8 1992-01-01 Australia 157916.081 NYGDPMKTPSAKD 17495000 SP.POP.TOTL

9 1992-04-01 Australia 159047.827 NYGDPMKTPSAKD 17495000 SP.POP.TOTL

10 1992-07-01 Australia 160658.176 NYGDPMKTPSAKD 17495000 SP.POP.TOTL

11 1992-09-01 Australia 163960.221 NYGDPMKTPSAKD 17495000 SP.POP.TOTL

12 1993-01-01 Australia 165097.495 NYGDPMKTPSAKD 17667000 SP.POP.TOTL

13 1993-04-01 Australia 166027.059 NYGDPMKTPSAKD 17667000 SP.POP.TOTL

14 1993-07-01 Australia 166203.179 NYGDPMKTPSAKD 17667000 SP.POP.TOTL

15 1993-09-01 Australia 169279.348 NYGDPMKTPSAKD 17667000 SP.POP.TOTL

16 1990-01-01 Sweden 79837.846 NYGDPMKTPSAKD 8558835 SP.POP.TOTL

17 1990-04-01 Sweden 80582.286 NYGDPMKTPSAKD 8558835 SP.POP.TOTL

18 1990-07-01 Sweden 79974.360 NYGDPMKTPSAKD 8558835 SP.POP.TOTL

19 1990-09-01 Sweden 80106.497 NYGDPMKTPSAKD 8558835 SP.POP.TOTL

20 1991-01-01 Sweden 79524.242 NYGDPMKTPSAKD 8617375 SP.POP.TOTL

21 1991-04-01 Sweden 79073.059 NYGDPMKTPSAKD 8617375 SP.POP.TOTL

22 1991-07-01 Sweden 79084.770 NYGDPMKTPSAKD 8617375 SP.POP.TOTL

23 1991-09-01 Sweden 79740.606 NYGDPMKTPSAKD 8617375 SP.POP.TOTL

24 1992-01-01 Sweden 79390.922 NYGDPMKTPSAKD 8668067 SP.POP.TOTL

25 1992-04-01 Sweden 79060.283 NYGDPMKTPSAKD 8668067 SP.POP.TOTL

26 1992-07-01 Sweden 78904.605 NYGDPMKTPSAKD 8668067 SP.POP.TOTL

27 1992-09-01 Sweden 76996.837 NYGDPMKTPSAKD 8668067 SP.POP.TOTL

28 1993-01-01 Sweden 75783.588 NYGDPMKTPSAKD 8718561 SP.POP.TOTL

29 1993-04-01 Sweden 76708.548 NYGDPMKTPSAKD 8718561 SP.POP.TOTL

30 1993-07-01 Sweden 77662.018 NYGDPMKTPSAKD 8718561 SP.POP.TOTL

31 1993-09-01 Sweden 77703.304 NYGDPMKTPSAKD 8718561 SP.POP.TOTL

Nice! When you merge on date first, the table is sorted by date then country. When forward fill is applied, Sweden's population value in January is used to fill in the missing values for both Australia and the Sweden for the remainder of the year. This is not what you want. The fill forward is using unintended data to fill in the missing values. However, when you merge on country first, the table is sorted by country then date, so the forward fill is applied appropriately in this situation.

**Daily XP350**

# Using merge\_asof()

**50 XP**

## 1. Using merge\_asof()

In this lesson we will talk about another method for ordered or time-series data called merge\_asof().

## 2. Using merge\_asof()

The merge\_asof() method is similar to an ordered left join. It has similar features as merge\_ordered(). However, unlike an ordered left join, merge\_asof() will match on the nearest value columns rather than equal values. This brings up an important point - whatever columns you merge on must be sorted. In the table shown here, when we merge on column "C", we bring back all of the rows from the left table.

## 3. Using merge\_asof()

However, the row selected from the right table is the last row whose "C" value is less than or equal to the "C" value in the left table. So, for example, the second row in the left table is matched with the third row in the right table. This because 3 is the closest value in the right table that is still less than or equal to 5.

## 4. Datasets

For this example, we will look at merging two tables. The first is stock price data for the Visa company with entries for every hour on Nov, 11, 2017. The second table is IBM stock prices on the same day with entries for roughly every five minutes.

## 5. merge\_asof() example

Let's use merge\_asof() to merge the tables. The input arguments are very similar to what we have already seen in the course. Here we list the left and right tables first. Then we define that we want to merge on the "date\_time" column. Finally, we provide a set of suffixes. Our output is similar to a left join, so we see all of the rows from the left Visa table. However, the values from the IBM table are based on how close the date\_time values match with the Visa table. Notice the first row and the IBM price of 149.11. Let's show the IBM table again and see why this value was chosen in the merger. It comes from the row indexed as 4. This row has the closest date\_time that is less than the date\_time in the Visa table. The next row has a date\_time that is slightly greater. We will adjust this behavior in our coming example.

## 6. merge\_asof() example with direction

This time in our merge\_asof() method, we list the direction argument as "forward". This will change the behavior of the method to select the first row in the right table whose "on" key column is greater than or equal to the left's key column. The default value for the direction argument is "backward". When we look at our results, we see different values for the IBM column. Let's again look at the first IBM value and trace it back to the IBM table. We see it in the row indexed as 5. Its date\_time is slightly greater than the date\_time in the visa table. Finally, you can set the direction argument to "nearest" which returns the nearest row in the right table regardless if it is forward or backwards.

## 7. When to use merge\_asof()

Now that we reviewed the merge\_asof() method, here are a couple of thoughts on when you might want to use it. First, you might think of this method when you are working with data sampled from a process and the dates or times may not exactly align. This is similar to what we did in our example. It could also be used when you are working on a time-series training set, where you do not want any events from the future to be visible before that point in time.

## 8. Let's practice!

Let's practice!

**Daily XP400**

**Exercise**

**Exercise**

**Using merge\_asof() to study stocks**

You have a feed of stock market prices that you record. You attempt to track the price every five minutes. Still, due to some network latency, the prices you record are roughly every 5 minutes. You pull your price logs for three banks, *JP Morgan* (JPM), *Wells Fargo* (WFC), and *Bank Of America* (BAC). You want to know how the price change of the two other banks compare to JP Morgan. Therefore, you will need to merge these three logs into one table. Afterward, you will use the pandas .diff() method to compute the price change over time. Finally, plot the price changes so you can review your analysis.

The three log files have been loaded for you as tables named jpm, wells, and bac.

**Instructions**

**100 XP**

* Use merge\_asof() to merge jpm (left table) and wells together on the date\_time column, where the rows with the ***nearest*** times are matched, and with suffixes=('', '\_wells'). Save to jpm\_wells.
* Use merge\_asof() to merge jpm\_wells (left table) and bac together on the date\_time column, where the rows with the closest times are matched, and with suffixes=('\_jpm', '\_bac'). Save to jpm\_wells\_bac.
* Using price\_diffs, create a line plot of the close price of JPM, WFC, and BAC only.
* # Use merge\_asof() to merge jpm and wells
* jpm\_wells = \_\_\_\_
* # Use merge\_asof() to merge jpm\_wells and bac
* jpm\_wells\_bac = \_\_\_\_
* # Compute price diff
* price\_diffs = jpm\_wells\_bac.diff()
* # Plot the price diff of the close of jpm, wells and bac only
* price\_diffs.plot(y=[\_\_\_\_, \_\_\_\_, \_\_\_\_])
* plt.show()

# Merge gdp and pop on country and date with fill date\_ctry = pd.merge\_ordered(gdp, pop, on=['country','date'], fill\_method='ffill') # Print date\_ctry print(date\_ctry)

# Use merge\_asof() to merge jpm and wells

jpm\_wells = pd.merge\_asof(jpm, wells, on='date\_time', suffixes=('', '\_wells'), direction= 'nearest')

# Use merge\_asof() to merge jpm\_wells and bac

jpm\_wells\_bac = pd.merge\_asof(jpm\_wells, bac, on=['date\_time'], suffixes=('\_jpm', '\_bac'), direction='nearest')

# Compute price diff

price\_diffs = jpm\_wells\_bac.diff()

print(price\_diffs)

# Plot the price diff of the close of jpm, wells and bac only

price\_diffs.plot(y=['close\_jpm', 'close\_wells', 'close\_bac'], kind='line')

plt.show()

# Use merge\_asof() to merge jpm and wells

jpm\_wells = pd.merge\_asof(jpm, wells, on='date\_time', suffixes=('', '\_wells'), direction= 'nearest')

# Use merge\_asof() to merge jpm\_wells and bac

jpm\_wells\_bac = pd.merge\_asof(jpm\_wells, bac, on=['date\_time'], suffixes=('\_jpm', '\_bac'), direction='nearest')

# Compute price diff

price\_diffs = jpm\_wells\_bac.diff()

print(price\_diffs)

# Plot the price diff of the close of jpm, wells and bac only

price\_diffs.plot(y=['close\_jpm', 'close\_wells', 'close\_bac'], kind='line')

plt.show()

date\_time close\_jpm close\_wells close\_bac

0 NaT NaN NaN NaN

1 0 days 00:04:47 0.060 -0.003 0.000

2 0 days 00:04:57 -0.449 -0.130 -0.164

3 0 days 00:05:54 0.009 -0.020 -0.010

4 0 days 00:04:05 0.075 0.014 0.005

5 0 days 00:05:30 0.205 0.081 0.069

6 0 days 00:04:37 -0.220 -0.065 -0.079

7 0 days 00:05:01 0.040 -0.045 0.015

8 0 days 00:05:03 -0.130 0.035 -0.019

9 0 days 00:05:18 0.050 0.015 0.019

10 0 days 00:04:56 0.060 0.025 0.079

11 0 days 00:05:28 0.130 -0.010 0.015

12 0 days 00:04:18 0.040 0.000 0.010

13 0 days 00:05:33 0.070 0.060 0.035

14 0 days 00:05:08 -0.010 -0.040 -0.005

15 0 days 00:04:45 0.060 -0.070 0.025

16 0 days 00:04:25 0.070 0.010 0.020

<script.py> output:

date\_time close\_jpm close\_wells close\_bac

0 NaT NaN NaN NaN

1 0 days 00:04:47 0.060 -0.003 0.000

2 0 days 00:04:57 -0.449 -0.130 -0.164

3 0 days 00:05:54 0.009 -0.020 -0.010

4 0 days 00:04:05 0.075 0.014 0.005

5 0 days 00:05:30 0.205 0.081 0.069

6 0 days 00:04:37 -0.220 -0.065 -0.079

7 0 days 00:05:01 0.040 -0.045 0.015

8 0 days 00:05:03 -0.130 0.035 -0.019

9 0 days 00:05:18 0.050 0.015 0.019

10 0 days 00:04:56 0.060 0.025 0.079

11 0 days 00:05:28 0.130 -0.010 0.015

12 0 days 00:04:18 0.040 0.000 0.010

13 0 days 00:05:33 0.070 0.060 0.035

14 0 days 00:05:08 -0.010 -0.040 -0.005

15 0 days 00:04:45 0.060 -0.070 0.025

16 0 days 00:04:25 0.070 0.010 0.020

Fabulous! You can see that during this period, the price change for these bank stocks was roughly the same, although the price change for \_JP Morgan\_ was more variable. The critical point here is that the merge\_asof() function is very useful in performing the fuzzy matching between the timestamps of all the tables.

**Daily XP470**

**Exercise**

**Exercise**

**Using merge\_asof() to create dataset**

The merge\_asof() function can be used to create datasets where you have a table of start and stop dates, and you want to use them to create a flag in another table. You have been given gdp, which is a table of quarterly GDP values of the US during the 1980s. Additionally, the table recession has been given to you. It holds the starting date of every US recession since 1980, and the date when the recession was declared to be over. Use merge\_asof() to merge the tables and create a status flag if a quarter was during a recession. Finally, to check your work, plot the data in a bar chart.

The tables gdp and recession have been loaded for you.

**Instructions**

**100 XP**

* Using merge\_asof(), merge gdp and recession on date, with gdp as the left table. Save to the variable gdp\_recession.
* Create a list using a list comprehension and a conditional expression, named is\_recession, where for each row if the gdp\_recession['econ\_status'] value is equal to 'recession' then enter 'r' else 'g'.
* Using gdp\_recession, plot a bar chart of gdp versus date, setting the color argument equal to is\_recession.
* # Merge gdp and recession on date using merge\_asof()
* gdp\_recession = \_\_\_\_
* # Create a list based on the row value of gdp\_recession['econ\_status']
* is\_recession = ['\_\_\_\_' if s=='recession' else '\_\_\_\_' for s in gdp\_recession['econ\_status']]
* # Plot a bar chart of gdp\_recession
* gdp\_recession.plot(kind=\_\_\_\_, y=\_\_\_\_, x=\_\_\_\_, color=\_\_\_\_, rot=90)
* plt.show()

# Use merge\_asof() to merge jpm and wells jpm\_wells = pd.merge\_asof(jpm, wells, on='date\_time', suffixes=('', '\_wells'), direction= 'nearest') # Use merge\_asof() to merge jpm\_wells and bac jpm\_wells\_bac = pd.merge\_asof(jpm\_wells, bac, on=['date\_time'], suffixes=('\_jpm', '\_bac'), direction='nearest') # Compute price diff price\_diffs = jpm\_wells\_bac.diff() print(price\_diffs) # Plot the price diff of the close of jpm, wells and bac only price\_diffs.plot(y=['close\_jpm', 'close\_wells', 'close\_bac'], kind='line') plt.show()# Merge gdp and recession on date using merge\_asof()

gdp\_recession = pd.merge\_asof(gdp, recession, on='date')

# Create a list based on the row value of gdp\_recession['econ\_status']

is\_recession = ['r' if s=='recession' else 'g' for s in gdp\_recession['econ\_status']]

# Plot a bar chart of gdp\_recession

gdp\_recession.plot(kind='bar', y='gdp', x='date', color=is\_recession, rot=90)

plt.show()

# Merge gdp and recession on date using merge\_asof()

gdp\_recession = pd.merge\_asof(gdp, recession, on='date')

# Create a list based on the row value of gdp\_recession['econ\_status']

is\_recession = ['r' if s=='recession' else 'g' for s in gdp\_recession['econ\_status']]

# Plot a bar chart of gdp\_recession

gdp\_recession.plot(kind='bar', y='date', x='gdp', color=is\_recession, rot=90)

plt.show()

# Merge gdp and recession on date using merge\_asof()

gdp\_recession = pd.merge\_asof(gdp, recession, on='date')

# Create a list based on the row value of gdp\_recession['econ\_status']

is\_recession = ['r' if s=='recession' else 'g' for s in gdp\_recession['econ\_status']]

# Plot a bar chart of gdp\_recession

gdp\_recession.plot(kind='bar', y='gdp', x='date', color=is\_recession, rot=90)

plt.show()

Terrific work! You can see from the chart that there were a number of quarters early in the 1980s where a recession was an issue. merge\_asof() allowed you to quickly add a flag to the gdp dataset by matching between two different dates, in one line of code! If you were to perform the same task using subsetting, it would have taken a lot more code.

**Daily XP570**

**Exercise**

**merge\_asof() and merge\_ordered() differences**

The merge\_asof() and merge\_ordered() functions are similar in the type of merge they perform and the input arguments they use. In this exercise, think about how the functions are different.

**Instructions**

**100XP**

* Drag and drop the statement into the appropriate box for either the merge\_asof() function, the merge\_ordered() function, or both if it applies to both functions.

+100 XP

Remarkable work! You were able to identify some of the similarities and differences between the functions. You are well on your way to mastering both!

Press enter to

**Daily XP670**

# Selecting data with .query()

**50 XP**

## 1. Selecting data with .query()

Now that you have learned quite a bit about combining data from different data sources, let's review a pandas method for selecting data from the table called the query() method. pandas provides many methods for selecting data, and query() is one of them.

## 2. The .query() method

The query() method accepts an input string that it will use to select rows to return from the table. For those familiar with SQL, the string you provide to the query function is similar to the portion after the WHERE clause of a SQL statement. However, don't let the SQL statement scare you, because prior knowledge of SQL isn't required. Let's look at an example.

## 3. Querying on a single condition

We have the following table named stocks with the stock price of Disney and Nike on different days. Now imagine we would like to select the rows where Nike is equal to or above 90. Here we provide a string to the query method. The string identifies that we want to condition which rows are returned by the value of the Nike column. We simply input "nike >= 90". The method returns all rows in stocks where Nike is greater than or equal to 90.

## 4. Querying on a multiple conditions, "and", "or"

Let's look at another example. Here we use the "and" keyword to select rows where Nike is greater than 90 and Disney is less than 140. The method returns two rows of data that match our criteria. Next, instead of using "and" we can also use the "or" keyword. This input string should select all rows where Nike is over 96 or Disney is less than 98. Now the function returns three rows that meet our criteria.

## 5. Updated dataset

Our next example shows that you can use the query method to select strings. Imagine now that we have an updated dataset, which is the stocks table in a slightly different format.

## 6. Using .query() to select text

We are interested in selecting all of the rows were the column stock equals "disney" or the column stock equals "nike" and close is less than 90. Let's pause here for a moment to look at our query string. Within the parentheses of our string, we check if the stock column is nike and the close column is less than 90. Both of these conditions have to be true for the parentheses section to return true. We then add that to the condition to check if stock is listed as "disney". When checking text, we use the double equal signs, similar to an if statement in Python. Also, when checking a text string, we used double quotes to surround the word. This is to avoid unintentionally ending our string statement since we used single quotes to start the statement. In our results, we see all of our Disney rows returned. Also, those rows were Nike is the stock name and the close price is less than 90 are returned.

## 7. Let's practice!

It's time for you to try using the query() method.

**Daily XP720**

**Exercise**

**Exercise**

**Explore financials with .query()**

You have been given a table of financial data from some popular social network companies called social\_fin. All of the values are in thousands of US dollars.

Use the .query() method and the IPython shell to explore social\_fin and select the **True** statement.

**Instructions**

**50 XP**

**Possible Answers**

* 

There 2 rows where the value is greater than $50,000,000K.

* 

There are 3 rows for total revenue for Facebook.

* 

There are 6 rows where the net income has a negative value.

* 

There are 45 rows, where the gross profit is greater than $100K.

# Merge gdp and recession on date using merge\_asof() gdp\_recession = pd.merge\_asof(gdp, recession, on='date') # Create a list based on the row value of gdp\_recession['econ\_status'] is\_recession = ['r' if s=='recession' else 'g' for s in gdp\_recession['econ\_status']] # Plot a bar chart of gdp\_recession gdp\_recession.plot(kind='bar', y='gdp', x='date', color=is\_recession, rot=90) plt.show()

**Daily XP50**

**Exercise**

**Exercise**

**Subsetting rows with .query()**

In this exercise, you will revisit GDP and population data for Australia and Sweden from the World Bank and expand on it using the .query() method. You'll merge the two tables and compute the GDP per capita. Afterwards, you'll use the .query() method to sub-select the rows and create a plot. Recall that you will need to merge on multiple columns in the proper order.

The tables gdp and pop have been loaded for you.

**Instructions 1/4**

* Use merge\_ordered() on gdp and pop on columns country and date with the fill feature, save to gdp\_pop and print.

# Merge gdp and pop on date and country with fill

gdp\_pop = \_\_\_\_

# Merge gdp and pop on date and country with fill

gdp\_pop = pd.merge\_ordered(gdp, pop, on=['country', 'date'], fill\_method='ffill')

print(gdp\_pop)

# Merge gdp and pop on date and country with fill

gdp\_pop = pd.merge\_ordered(gdp, pop, on=['country', 'date'], fill\_method='ffill')

# Merge gdp and pop on date and country with fill

gdp\_pop = pd.merge\_ordered(gdp, pop, on=['country', 'date'], fill\_method='ffill')

print(gdp\_pop)

ERROR! Session/line number was not unique in database. History logging moved to new session 5

date country gdp series\_code\_x pop series\_code\_y

0 1990-01-01 Australia 158051.132 NYGDPMKTPSAKD 17065100 SP.POP.TOTL

1 1990-04-01 Australia 158263.582 NYGDPMKTPSAKD 17065100 SP.POP.TOTL

2 1990-07-01 Australia 157329.279 NYGDPMKTPSAKD 17065100 SP.POP.TOTL

3 1990-09-01 Australia 158240.678 NYGDPMKTPSAKD 17065100 SP.POP.TOTL

4 1991-01-01 Australia 156195.954 NYGDPMKTPSAKD 17284000 SP.POP.TOTL

5 1991-04-01 Australia 155989.033 NYGDPMKTPSAKD 17284000 SP.POP.TOTL

6 1991-07-01 Australia 156635.858 NYGDPMKTPSAKD 17284000 SP.POP.TOTL

7 1991-09-01 Australia 156744.057 NYGDPMKTPSAKD 17284000 SP.POP.TOTL

8 1992-01-01 Australia 157916.081 NYGDPMKTPSAKD 17495000 SP.POP.TOTL

9 1992-04-01 Australia 159047.827 NYGDPMKTPSAKD 17495000 SP.POP.TOTL

10 1992-07-01 Australia 160658.176 NYGDPMKTPSAKD 17495000 SP.POP.TOTL

11 1992-09-01 Australia 163960.221 NYGDPMKTPSAKD 17495000 SP.POP.TOTL

12 1993-01-01 Australia 165097.495 NYGDPMKTPSAKD 17667000 SP.POP.TOTL

13 1993-04-01 Australia 166027.059 NYGDPMKTPSAKD 17667000 SP.POP.TOTL

14 1993-07-01 Australia 166203.179 NYGDPMKTPSAKD 17667000 SP.POP.TOTL

15 1993-09-01 Australia 169279.348 NYGDPMKTPSAKD 17667000 SP.POP.TOTL

16 1990-01-01 Sweden 79837.846 NYGDPMKTPSAKD 8558835 SP.POP.TOTL

17 1990-04-01 Sweden 80582.286 NYGDPMKTPSAKD 8558835 SP.POP.TOTL

18 1990-07-01 Sweden 79974.360 NYGDPMKTPSAKD 8558835 SP.POP.TOTL

19 1990-09-01 Sweden 80106.497 NYGDPMKTPSAKD 8558835 SP.POP.TOTL

20 1991-01-01 Sweden 79524.242 NYGDPMKTPSAKD 8617375 SP.POP.TOTL

21 1991-04-01 Sweden 79073.059 NYGDPMKTPSAKD 8617375 SP.POP.TOTL

22 1991-07-01 Sweden 79084.770 NYGDPMKTPSAKD 8617375 SP.POP.TOTL

23 1991-09-01 Sweden 79740.606 NYGDPMKTPSAKD 8617375 SP.POP.TOTL

24 1992-01-01 Sweden 79390.922 NYGDPMKTPSAKD 8668067 SP.POP.TOTL

25 1992-04-01 Sweden 79060.283 NYGDPMKTPSAKD 8668067 SP.POP.TOTL

26 1992-07-01 Sweden 78904.605 NYGDPMKTPSAKD 8668067 SP.POP.TOTL

27 1992-09-01 Sweden 76996.837 NYGDPMKTPSAKD 8668067 SP.POP.TOTL

28 1993-01-01 Sweden 75783.588 NYGDPMKTPSAKD 8718561 SP.POP.TOTL

29 1993-04-01 Sweden 76708.548 NYGDPMKTPSAKD 8718561 SP.POP.TOTL

30 1993-07-01 Sweden 77662.018 NYGDPMKTPSAKD 8718561 SP.POP.TOTL

31 1993-09-01 Sweden 77703.304 NYGDPMKTPSAKD 8718561 SP.POP.TOTL

Add a column named gdp\_per\_capita to gdp\_pop that divides gdp by pop.

# Merge gdp and pop on date and country with fill

gdp\_pop = pd.merge\_ordered(gdp, pop, on=['country','date'], fill\_method='ffill')

# Add a column named gdp\_per\_capita to gdp\_pop that divides the gdp by pop

\_\_\_

# Merge gdp and pop on date and country with fill

gdp\_pop = pd.merge\_ordered(gdp, pop, on=['country','date'], fill\_method='ffill')

# Add a column named gdp\_per\_capita to gdp\_pop that divides the gdp by pop

\_\_\_

# Merge gdp and pop on date and country with fill

gdp\_pop = pd.merge\_ordered(gdp, pop, on=['country','date'], fill\_method='ffill')

# Add a column named gdp\_per\_capita to gdp\_pop that divides the gdp by pop

gdp\_pop['gdp\_per\_capita'] = gdp\_pop['gdp'] / gdp\_pop['pop']

# Pivot table of gdp\_per\_capita, where index is date and columns is country

gdp\_pivot = gdp\_pop.pivot\_table('gdp\_per\_capita', '\_\_\_\_', '\_\_\_\_')

# Merge gdp and pop on date and country with fill gdp\_pop = pd.merge\_ordered(gdp, pop, on=['country','date'], fill\_method='ffill') # Add a column named gdp\_per\_capita to gdp\_pop that divides the gdp by pop gdp\_pop['gdp\_per\_capita'] = gdp\_pop['gdp'] /gdp\_pop['pop']

gdp\_pop = pd.merge\_ordered(gdp, pop, on=['country','date'], fill\_method='ffill')

# Add a column named gdp\_per\_capita to gdp\_pop that divides the gdp by pop

gdp\_pop['gdp\_per\_capita'] = gdp\_pop['gdp'] / gdp\_pop['pop']

# Pivot table of gdp\_per\_capita, where index is date and columns is country

gdp\_pivot = gdp\_pop.pivot\_table('gdp\_per\_capita', index='date', columns='country')

# Merge gdp and pop on date and country with fill

gdp\_pop = pd.merge\_ordered(gdp, pop, on=['country','date'], fill\_method='ffill')

# Add a column named gdp\_per\_capita to gdp\_pop that divides the gdp by pop

gdp\_pop['gdp\_per\_capita'] = gdp\_pop['gdp'] /gdp\_pop['pop']

# Merge gdp and pop on date and country with fill

gdp\_pop = pd.merge\_ordered(gdp, pop, on=['country','date'], fill\_method='ffill')

# Add a column named gdp\_per\_capita to gdp\_pop that divides the gdp by pop

gdp\_pop['gdp\_per\_capita'] = gdp\_pop['gdp'] / gdp\_pop['pop']

# Pivot table of gdp\_per\_capita, where index is date and columns is country

gdp\_pivot = gdp\_pop.pivot\_table('gdp\_per\_capita', index='date', columns='country')

# Merge gdp and pop on date and country with fill

gdp\_pop = pd.merge\_ordered(gdp, pop, on=['country','date'], fill\_method='ffill')

# Add a column named gdp\_per\_capita to gdp\_pop that divides the gdp by pop

gdp\_pop['gdp\_per\_capita'] = gdp\_pop['gdp'] /gdp\_pop['pop']

# Merge gdp and pop on date and country with fill

gdp\_pop = pd.merge\_ordered(gdp, pop, on=['country','date'], fill\_method='ffill')

# Add a column named gdp\_per\_capita to gdp\_pop that divides the gdp by pop

gdp\_pop['gdp\_per\_capita'] = gdp\_pop['gdp'] / gdp\_pop['pop']

# Pivot table of gdp\_per\_capita, where index is date and columns is country

gdp\_pivot = gdp\_pop.pivot\_table('gdp\_per\_capita', index='date', columns='country')

# Merge gdp and pop on date and country with fill

gdp\_pop = pd.merge\_ordered(gdp, pop, on=['country','date'], fill\_method='ffill')

# Add a column named gdp\_per\_capita to gdp\_pop that divides the gdp by pop

gdp\_pop['gdp\_per\_capita'] = gdp\_pop['gdp'] / gdp\_pop['pop']

# Pivot data so gdp\_per\_capita, where index is date and columns is country

gdp\_pivot = gdp\_pop.pivot\_table('gdp\_per\_capita', 'date', 'country')

# Select dates equal to or greater than 1991-01-01

recent\_gdp\_pop = gdp\_pivot.query('\_\_\_\_')

# Plot recent\_gdp\_pop

recent\_gdp\_pop.plot(rot=90)

plt.show()

# Merge gdp and pop on date and country with fill

gdp\_pop = pd.merge\_ordered(gdp, pop, on=['country','date'], fill\_method='ffill')

# Add a column named gdp\_per\_capita to gdp\_pop that divides the gdp by pop

gdp\_pop['gdp\_per\_capita'] = gdp\_pop['gdp'] / gdp\_pop['pop']

# Pivot data so gdp\_per\_capita, where index is date and columns is country

gdp\_pivot = gdp\_pop.pivot\_table('gdp\_per\_capita', 'date', 'country')

# Select dates equal to or greater than 1991-01-01

recent\_gdp\_pop = gdp\_pivot.query('date >= "1991-01-01"')

# Plot recent\_gdp\_pop

recent\_gdp\_pop.plot(rot=90)

plt.show()

Use .query() to select rows from gdp\_pivot where date is greater than equal to "1991-01-01". Save as recent\_gdp\_pop.

# Merge gdp and pop on date and country with fill gdp\_pop = pd.merge\_ordered(gdp, pop, on=['country','date'], fill\_method='ffill') # Add a column named gdp\_per\_capita to gdp\_pop that divides the gdp by pop gdp\_pop['gdp\_per\_capita'] = gdp\_pop['gdp'] / gdp\_pop['pop'] # Pivot data so gdp\_per\_capita, where index is date and columns is country gdp\_pivot = gdp\_pop.pivot\_table('gdp\_per\_capita', 'date', 'country') # Select dates equal to or greater than 1991-01-01 recent\_gdp\_pop = gdp\_pivot.query('date >= "1991-01-01"') # Plot recent\_gdp\_pop recent\_gdp\_pop.plot(rot=90) plt.show()

Amazing! You can see from the plot that the per capita GDP of Australia passed Sweden in 1992. By using the .query() method, you were able to select the appropriate rows easily. The .query() method is easy to read and straightforward.

**Daily XP150**

# Reshaping data with .melt()

**50 XP**

## 1. Reshaping data with .melt()

In our last lesson of the course, let's talk about the melt method. This method will unpivot a table from wide to long format. This is often a much more computer-friendly format, therefore making this a valuable method to know.

## 2. Wide versus long data

Sometimes we will come across data where every row relates to one subject, and each column has different information about an attribute of that subject. Data formatted in this way is often called wide. There are other times when the information about one subject is found over many rows, and each row has one attribute about that subject. Data formatted in this way is often called long or tall. In general, wide formatted data is easier to read by people than long formatted. However, long formatted data is often more accessible for computers to work with.

## 3. What does the .melt() method do?

The melt method will allow us to unpivot, or change the format of, our dataset. In this image, we change the height and weight columns from their wide horizontal placement to a long vertical placement.

## 4. Dataset in wide format

To demonstrate the melt method, let's start with this dataset of financial metrics of two popular social media companies. Notice that the years are horizontal. Let's change them so that they are vertically placed.

## 5. Example of .melt()

Here we call the melt() method on the table social\_fin. The first input argument to the method is id\_vars. These are columns to be used as identifier variables. We can also think of them as columns in our original dataset that we do not want to change. In our output, we print the first ten rows. Our years are listed vertically. Our final column now has all of our values in one column versus multiple columns. Again, this is a much more computer-friendly format than our original table. We unpivoted each of the separate columns 2016 through 2019. Our output has data for every year in our starting table, but again, we are only showing the first couple of rows. In the next example, we will look at how to control what columns are unpivoted.

## 6. Melting with value\_vars

This time, let's use the argument value\_vars with the melt() method. This argument will allow us to control which columns are unpivoted. Here, we unpivot only the 2018 and 2017 columns. Our output now only has data for the years 2018 and 2017. Additionally, the order of the value\_var was kept. The output starts with 2018, then moves to 2017. Finally, notice that the column with the years is now named variable, and our values column is named value. We will adjust that in our next example.

## 7. Melting with column names

In this example, we have added some additional inputs to our melt() method. The var\_name argument will allow us to set the name of the year column in the output. Similarly, the value\_name argument will allow us to set the name of the value column in the output. We again print the first few rows of the output. It is the same as before, except our variable and value columns are renamed year and dollars, respectively. We have seen how the melt() method is useful for reshaping our tables. Imagine a situation where you have merged many columns, making your table very wide. The merge() method can then be used to reshape that table into a more computer-friendly format.

## 8. Let's practice!

Alright, time to practice!

**Daily XP200**

**Exercise**

**Exercise**

**Select the right .melt() arguments**

You are given a table named inflation. Chose the option to get the ***same*** output as the table below.

country indicator year annual

0 Brazil Inflation % 2017 3.45

1 Canada Inflation % 2017 1.60

2 France Inflation % 2017 1.03

3 India Inflation % 2017 2.49

4 Brazil Inflation % 2018 3.66

5 Canada Inflation % 2018 2.27

6 France Inflation % 2018 1.85

7 India Inflation % 2018 4.86

8 Brazil Inflation % 2019 3.73

9 Canada Inflation % 2019 1.95

10 France Inflation % 2019 1.11

11 India Inflation % 2019 7.66

**Instructions**

**50 XP**

**Possible Answers**

* 

inflation.melt(id\_vars=['country','indicator'], var\_name='annual')

* 

inflation.melt(id\_vars=['country'], var\_name='indicator', value\_name='annual')

* 

inflation.melt(id\_vars=['country','indicator'], var\_name='year', value\_name='annual')

* 

inflation.melt(id\_vars=['country'], var\_name='year', value\_name='annual')

# Merge gdp and pop on date and country with fill gdp\_pop = pd.merge\_ordered(gdp, pop, on=['country','date'], fill\_method='ffill') # Add a column named gdp\_per\_capita to gdp\_pop that divides the gdp by pop gdp\_pop['gdp\_per\_capita'] = gdp\_pop['gdp'] / gdp\_pop['pop'] # Pivot data so gdp\_per\_capita, where index is date and columns is country gdp\_pivot = gdp\_pop.pivot\_table('gdp\_per\_capita', 'date', 'country') # Select dates equal to or greater than 1991-01-01 recent\_gdp\_pop = gdp\_pivot.query('date >= "1991-01-01"') # Plot recent\_gdp\_pop recent\_gdp\_pop.plot(rot=90) plt.show()

##### Possible Answers

* 

inflation.melt(id\_vars=['country','indicator'], var\_name='annual')

* 

inflation.melt(id\_vars=['country'], var\_name='indicator', value\_name='annual')

* 

**inflation.melt(id\_vars=['country','indicator'], var\_name='year', value\_name='annual') This is the answer**

* 

inflation.melt(id\_vars=['country'], var\_name='year', value\_name='annual')

country indicator year annual 0 Brazil Inflation % 2017 3.45 1 Canada Inflation % 2017 1.60 2 France Inflation % 2017 1.03 3 India Inflation % 2017 2.49 4 Brazil Inflation % 2018 3.66 5 Canada Inflation % 2018 2.27 6 France Inflation % 2018 1.85 7 India Inflation % 2018 4.86 8 Brazil Inflation % 2019 3.73 9 Canada Inflation % 2019 1.95 10 France Inflation % 2019 1.11 11 India Inflation % 2019 7.66

In [1]:

inflation.melt(id\_vars=['country','indicator'], var\_name='year', value\_name='annual')

Out[1]:

country indicator year annual

0 Brazil Inflation % 2017 3.45

1 Canada Inflation % 2017 1.60

2 France Inflation % 2017 1.03

3 India Inflation % 2017 2.49

4 Brazil Inflation % 2018 3.66

5 Canada Inflation % 2018 2.27

6 France Inflation % 2018 1.85

7 India Inflation % 2018 4.86

8 Brazil Inflation % 2019 3.73

9 Canada Inflation % 2019 1.95

10 France Inflation % 2019 1.11

11 India Inflation % 2019 7.66

Magnificent! You identified the correct values to pass to the id\_vars argument. These columns are not unpivoted. Finally, the other arguments set the name for the year and value columns.

**Daily XP250**

**Exercise**

**Exercise**

**Using .melt() to reshape government data**

The US Bureau of Labor Statistics (BLS) often provides data series in an easy-to-read format - it has a separate column for each month, and each year is a different row. Unfortunately, this wide format makes it difficult to plot this information over time. In this exercise, you will reshape a table of US unemployment rate data from the BLS into a form you can plot using .melt(). You will need to add a date column to the table and sort by it to plot the data correctly.

The unemployment rate data has been loaded for you in a table called ur\_wide. You are encouraged to view the table in the IPython shell before beginning the exercise.

**Instructions**

**100 XP**

* Use .melt() to unpivot all of the columns of ur\_wide except year and ensure that the columns with the months and values are named month and unempl\_rate, respectively. Save the result as ur\_tall.
* Add a column to ur\_tall named date which combines the year and month columns as *year*-*month* format into a larger string, and converts it to a date data type.
* Sort ur\_tall by date and save as ur\_sorted.
* Using ur\_sorted, plot unempl\_rate on the y-axis and date on the x-axis.
* # unpivot everything besides the year column
* ur\_tall = \_\_\_\_
* # Create a date column using the month and year columns of ur\_tall
* ur\_tall['date'] = pd.to\_datetime(ur\_tall['\_\_\_\_'] + '-' + \_\_\_\_)
* # Sort ur\_tall by date in ascending order
* ur\_sorted = \_\_\_\_
* # Plot the unempl\_rate by date
* ur\_sorted.plot(\_\_\_\_)
* plt.show()

# unpivot everything besides the year column

ur\_tall = ur\_wide.melt(id\_vars='year', var\_name='month', value\_name='unempl\_rate')

# Create a date column using the month and year columns of ur\_tall

ur\_tall['date'] = pd.to\_datetime(ur\_tall['year'] + '-' + ur\_tall['month'])

# Sort ur\_tall by date in ascending order

ur\_sorted = ur\_tall.sort\_values('date')

# Plot the unempl\_rate by date

ur\_sorted.plot(y='unempl\_rate', x='date', kind='line')

plt.show()

# unpivot everything besides the year column ur\_tall = ur\_wide.melt(id\_vars='year', var\_name='month', value\_name='unempl\_rate') # Create a date column using the month and year columns of ur\_tall ur\_tall['date'] = pd.to\_datetime(ur\_tall['year'] + '-' + ur\_tall['month']) # Sort ur\_tall by date in ascending order ur\_sorted = ur\_tall.sort\_values('date') # Plot the unempl\_rate by date ur\_sorted.plot(y='unempl\_rate', x='date', kind='line') plt.show()

Nice going! The plot shows a steady decrease in the unemployment rate with an increase near the end. This increase is likely the effect of the COVID-19 pandemic and its impact on shutting down most of the US economy. In general, data is often provided (\_especially by governments\_) in a format that is easily read by people but not by machines. The .melt() method is a handy tool for reshaping data into a useful form.

**Daily XP350**

**Exercise**

**Exercise**

**Using .melt() for stocks vs bond performance**

It is widespread knowledge that the price of bonds is inversely related to the price of stocks. In this last exercise, you'll review many of the topics in this chapter to confirm this. You have been given a table of percent change of the US 10-year treasury bond price. It is in a wide format where there is a separate column for each year. You will need to use the .melt() method to reshape this table.

Additionally, you will use the .query() method to filter out unneeded data. You will merge this table with a table of the percent change of the Dow Jones Industrial stock index price. Finally, you will plot data.

The tables ten\_yr and dji have been loaded for you.

**Instructions**

**100 XP**

* Use .melt() on ten\_yr to unpivot everything except the metric column, setting var\_name='date' and value\_name='close'. Save the result to bond\_perc.
* Using the .query() method, select only those rows were metric equals 'close', and save to bond\_perc\_close.
* Use merge\_ordered() to merge dji (left table) and bond\_perc\_close on date with an inner join, and set suffixes equal to ('\_dow', '\_bond'). Save the result to dow\_bond.
* Using dow\_bond, plot only the Dow and bond values.

# Use melt on ten\_yr, unpivot everything besides the metric column

bond\_perc = \_\_\_\_

# Use query on bond\_perc to select only the rows where metric=close

bond\_perc\_close = \_\_\_\_

# Merge (ordered) dji and bond\_perc\_close on date with an inner join

dow\_bond = \_\_\_\_

# Plot only the close\_dow and close\_bond columns

dow\_bond.plot(\_\_\_\_, x='date', rot=90)

plt.show()

# unpivot everything besides the year column ur\_tall = ur\_wide.melt(id\_vars='year', var\_name='month', value\_name='unempl\_rate') # Create a date column using the month and year columns of ur\_tall ur\_tall['date'] = pd.to\_datetime(ur\_tall['year'] + '-' + ur\_tall['month']) # Sort ur\_tall by date in ascending order ur\_sorted = ur\_tall.sort\_values('date') # Plot the unempl\_rate by date ur\_sorted.plot(y='unempl\_rate', x='date', kind='line') plt.show()

In [1]:

print(ten\_yr)

metric 2007-02-01 2007-03-01 2007-04-01 2007-05-01 ... 2009-08-01 2009-09-01 2009-10-01 2009-11-01 2009-12-01

0 open 0.033 -0.060 0.025 -0.004 ... -0.007 -0.047 -0.032 0.034 -0.051

1 high -0.007 -0.041 0.022 0.031 ... 0.032 -0.090 0.012 -0.004 0.099

2 low -0.016 -0.008 0.031 -0.002 ... 0.040 -0.036 -0.051 0.030 0.007

3 close -0.057 0.022 -0.004 0.056 ... -0.029 -0.028 0.026 -0.056 0.201

[4 rows x 36 columns]

In [2]:

print(dji)

date close

0 2007-02-01 0.005

1 2007-03-01 -0.026

2 2007-04-01 0.049

3 2007-05-01 0.052

4 2007-06-01 -0.016

.. ... ...

154 2019-12-01 NaN

155 2020-01-01 NaN

156 2020-02-01 -0.010

157 2020-03-01 -0.216

158 2020-04-01 0.035

[159 rows x 2 columns]

# Use melt on ten\_yr, unpivot everything besides the metric column

bond\_perc = ten\_yr.melt(id\_vars='metric', var\_name='date', value\_name='close')

# Use query on bond\_perc to select only the rows where metric=close

bond\_perc\_close = bond\_perc.query('metric == "close"')

# Merge (ordered) dji and bond\_perc\_close on date with an inner join

dow\_bond = pd.merge\_ordered(dji, bond\_perc\_close, on='date', how='inner', suffixes=('\_dow', '\_bond'))

# Plot only the close\_dow and close\_bond columns

dow\_bond.plot(\_\_\_\_, x='date', rot=90)

plt.show()

# Use melt on ten\_yr, unpivot everything besides the metric column

bond\_perc = ten\_yr.melt(id\_vars='metric', var\_name='date', value\_name='close')

# Use query on bond\_perc to select only the rows where metric=close

bond\_perc\_close = bond\_perc.query('metric == "close"')

# Merge (ordered) dji and bond\_perc\_close on date with an inner join

dow\_bond = pd.merge\_ordered(dji, bond\_perc\_close, on='date', how='inner', suffixes=('\_dow', '\_bond'))

# Plot only the close\_dow and close\_bond columns

dow\_bond.plot(y= ['close\_dow','close\_bond'], x='date', rot=90)

plt.show()

# Use melt on ten\_yr, unpivot everything besides the metric column

bond\_perc = ten\_yr.melt(id\_vars='metric', var\_name='date', value\_name='close')

# Use query on bond\_perc to select only the rows where metric=close

bond\_perc\_close = bond\_perc.query('metric == "close"')

# Merge (ordered) dji and bond\_perc\_close on date with an inner join

dow\_bond = pd.merge\_ordered(dji, bond\_perc\_close, on='date', how='inner', suffixes=('\_dow', '\_bond'))

# Plot only the close\_dow and close\_bond columns

dow\_bond.plot(y= ['close\_dow','close\_bond'], x='date', rot=90)

plt.show()

In [4]:

print(dji)

date close

0 2007-02-01 0.005

1 2007-03-01 -0.026

2 2007-04-01 0.049

3 2007-05-01 0.052

4 2007-06-01 -0.016

.. ... ...

154 2019-12-01 NaN

155 2020-01-01 NaN

156 2020-02-01 -0.010

157 2020-03-01 -0.216

158 2020-04-01 0.035

[159 rows x 2 columns]

# Use melt on ten\_yr, unpivot everything besides the metric column

bond\_perc = ten\_yr.melt(id\_vars='metric', var\_name='date', value\_name='close')

# Use query on bond\_perc to select only the rows where metric=close

bond\_perc\_close = bond\_perc.query('metric == "close"')

# Merge (ordered) dji and bond\_perc\_close on date with an inner join

dow\_bond = pd.merge\_ordered(dji, bond\_perc\_close, on='date', how='inner', suffixes=('\_dow', '\_bond'))

# Plot only the close\_dow and close\_bond columns

dow\_bond.plot(y= ['close\_dow','close\_bond'], x='date', rot=90)

plt.show()

In [5]:

print(dow\_bond)

date close\_dow metric close\_bond

0 2007-02-01 0.005 close -0.057

1 2007-03-01 -0.026 close 0.022

2 2007-04-01 0.049 close -0.004

3 2007-05-01 0.052 close 0.056

4 2007-06-01 -0.016 close 0.029

5 2007-07-01 0.038 close -0.052

6 2007-08-01 -0.064 close -0.049

7 2007-09-01 0.067 close 0.009

8 2007-10-01 0.002 close -0.023

9 2007-11-01 -0.024 close -0.112

10 2007-12-01 -0.011 close 0.016

11 2008-01-01 -0.059 close -0.098

12 2008-02-01 -0.036 close -0.029

13 2008-03-01 0.013 close -0.029

14 2008-04-01 0.021 close 0.095

15 2008-05-01 -0.001 close 0.076

16 2008-06-01 -0.043 close -0.017

17 2008-07-01 -0.057 close 0.000

18 2008-08-01 0.025 close -0.042

19 2008-09-01 -0.069 close 0.004

20 2008-10-01 -0.154 close 0.037

21 2008-11-01 -0.080 close -0.255

22 2008-12-01 0.058 close -0.241

23 2009-01-01 -0.037 close 0.267

24 2009-02-01 -0.165 close 0.069

25 2009-03-01 0.042 close -0.117

26 2009-04-01 0.065 close 0.164

27 2009-05-01 0.057 close 0.109

28 2009-06-01 0.039 close 0.017

29 2009-07-01 -0.048 close -0.006

30 2009-08-01 0.111 close -0.029

31 2009-09-01 0.058 close -0.028

32 2009-10-01 -0.008 close 0.026

33 2009-11-01 0.077 close -0.056

34 2009-12-01 -0.003 close 0.201

Super job! You used many of the techniques we have reviewed in this chapter to produce the plot. The plot confirms that the bond and stock prices are inversely correlated. Often as the price of stocks increases, the price for bonds decreases.

In [4]:

print(dji)

date close

0 2007-02-01 0.005

1 2007-03-01 -0.026

2 2007-04-01 0.049

3 2007-05-01 0.052

4 2007-06-01 -0.016

.. ... ...

154 2019-12-01 NaN

155 2020-01-01 NaN

156 2020-02-01 -0.010

157 2020-03-01 -0.216

158 2020-04-01 0.035

[159 rows x 2 columns]

# Use melt on ten\_yr, unpivot everything besides the metric column

bond\_perc = ten\_yr.melt(id\_vars='metric', var\_name='date', value\_name='close')

# Use query on bond\_perc to select only the rows where metric=close

bond\_perc\_close = bond\_perc.query('metric == "close"')

# Merge (ordered) dji and bond\_perc\_close on date with an inner join

dow\_bond = pd.merge\_ordered(dji, bond\_perc\_close, on='date', how='inner', suffixes=('\_dow', '\_bond'))

# Plot only the close\_dow and close\_bond columns

dow\_bond.plot(y= ['close\_dow','close\_bond'], x='date', rot=90)

plt.show()

In [5]:

print(dow\_bond)

date close\_dow metric close\_bond

0 2007-02-01 0.005 close -0.057

1 2007-03-01 -0.026 close 0.022

2 2007-04-01 0.049 close -0.004

3 2007-05-01 0.052 close 0.056

4 2007-06-01 -0.016 close 0.029

5 2007-07-01 0.038 close -0.052

6 2007-08-01 -0.064 close -0.049

7 2007-09-01 0.067 close 0.009

8 2007-10-01 0.002 close -0.023

9 2007-11-01 -0.024 close -0.112

10 2007-12-01 -0.011 close 0.016

11 2008-01-01 -0.059 close -0.098

12 2008-02-01 -0.036 close -0.029

13 2008-03-01 0.013 close -0.029

14 2008-04-01 0.021 close 0.095

15 2008-05-01 -0.001 close 0.076

16 2008-06-01 -0.043 close -0.017

17 2008-07-01 -0.057 close 0.000

18 2008-08-01 0.025 close -0.042

19 2008-09-01 -0.069 close 0.004

20 2008-10-01 -0.154 close 0.037

21 2008-11-01 -0.080 close -0.255

22 2008-12-01 0.058 close -0.241

23 2009-01-01 -0.037 close 0.267

24 2009-02-01 -0.165 close 0.069

25 2009-03-01 0.042 close -0.117

26 2009-04-01 0.065 close 0.164

27 2009-05-01 0.057 close 0.109

28 2009-06-01 0.039 close 0.017

29 2009-07-01 -0.048 close -0.006

30 2009-08-01 0.111 close -0.029

31 2009-09-01 0.058 close -0.028

32 2009-10-01 -0.008 close 0.026

33 2009-11-01 0.077 close -0.056

34 2009-12-01 -0.003 close 0.201

## 1. Course wrap-up

Congratulations! You have completed all the videos and exercises. You should feel proud.

## 2. You're this high performance race car now

It wasn't long ago when you started the course. Now you're like this high-performance race car when it comes to combining data. In this course, we covered a lot of topics.

1. 1 Photo by jae park from Pexels

## 3. Data merging basics

In chapter one, you performed an inner join with the default settings using the merge() method. You also learned about the different types of table relationships and merging multiple tables. These are the basics of using the combining data with the merge() method.

## 4. Merging tables with different join types

In the second chapter, you expanded your joining skills to many different types of joins. You also learned how to merge a table to itself and how to merge using indexes.

## 5. Advanced merging and concatenating

In the third chapter, you learned about filtering joins, such as semi- and anti joins. You also learned how to combine data vertically with the concat() method and about the importance of data integrity.

## 6. Merging ordered and time-series data

In the last chapter you discovered how to merge ordered and time-series data with merge\_ordered() and merge\_asof(). In the last lesson, you learned how to use melt().

## 7. Thank you!