|  |
| --- |
| **W6 – Joining Data with Pandas** |

Save your W6 notebook with the following naming conventions.

ID\_Name\_SecNo\_W6.ipynb,

for example

**6113333\_JohnWick\_541\_W6.ipynb**

**Joining Data**

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 worksheet will focus on the vital skill of merging tables together.

As an example, we are considering wards data of Chicago city.

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.

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer screen

Description automatically generated**

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.

A screenshot of a computer

Description automatically generated

The pandas package has an excellent DataFrame method for performing this type of merging 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”.

A screenshot of a computer

Description automatically generated A diagram of a diagram

Description automatically generated

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.

**A screenshot of a computer program

Description automatically generated**

# Let’s Practice. 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.

Load taxi\_owners.p to taxi\_owners and load taxi\_vehicles.p to taxi\_vehicles.

1. **Merge taxi\_owners with taxi\_veh on the column vid, and save the result to taxi\_own\_veh.**

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

**Select the fuel\_type column from taxi\_own\_veh and print the value\_counts() to find the most popular fuel\_types used.**

**Answer: Hybrid**

# 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

1. **Wards\_Offices\_Altered.csv is available which contains only 46 wards. Load this .csv to wards\_altered and merge it with census on the column ward. Save the result to wards\_census\_altered and observe the number of rows.**

**N-to-N Relationships**

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

A diagram of a table

Description automatically generated with medium confidence A chart with green squares and blue arrows

Description automatically generated

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.

Back to wards data.

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.

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer screen

Description automatically generated

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.

A screenshot of a computer code

Description automatically generated

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.

A white rectangular object with a black border

Description automatically generated

# 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)

Load licenses and biz\_owners DataFrames from licenses.p and business\_owners.p, respectively.

1. **Complete the following tasks.**

* 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 using .agg({'account':'count'}). 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.

**Merging multiple tables.**

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

A chart with green squares and arrows

Description automatically generated with medium confidence

**Note that you cannot try the following codes since one of the data set is not available. But you can try to merge multiple tables in the exercises.**

A screen shot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

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.

A screenshot of a computer screen

Description automatically generated

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.

A screenshot of a computer program

Description automatically generated

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.

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer code

Description automatically generated

A graph with blue lines

Description automatically generated with medium confidence

**Let’s Practice.**

# 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.

Load cta\_calendar.p, cta\_ridership.p and stations.p to cal, ridership, and stations DataFrames, respectively. The relationship between the tables can be seen in the diagram below.

A close-up of a list of rides

Description automatically generated

1. **Complete the following tasks.**

* Merge the ridership and cal tables together, starting with the ridership table on the left and save the result to the variable ridership\_cal.
* Extend the previous merge to three tables by also merging the stations table.
* Create a variable called filter\_criteria to select the appropriate rows from the merged table so that you can sum the rides column.

Answer: 140005

# 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 are to be loaded from licenses.p, ward.p and zip\_demo.p, respectively.

1. **Complete the following tasks.**

* 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 (agg({'income':'median'}).

**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.

A diagram of a left and right data frame

Description automatically generated A table with arrows and letters

Description automatically generated

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.

To understand better, let’s explore new data sets.

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.

A screenshot of a computer

Description automatically generated

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.

A screenshot of a computer

Description automatically generated

To merge these two tables with a left join, we use our merge method similar to what we learned previously. 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 earlier 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.

A screenshot of a computer program

Description automatically generated

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.

A white and black rectangle

Description automatically generated

**Let’s Practice.**

# 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 are to be loaded from movies.p and financials.p

1. **Complete the following tasks.**

* What column is likely the best column to merge the two tables on?
* Merge the movies table, as the left table, with the financials table using a left join, and save the result to movies\_financials.
* Count the number of rows in movies\_financials with a null value in the budget column.

Answer: 1574

# 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 are to be loaded from toy\_story.csv and taglines.p, respectively.

1. **Complete the following tasks.**

* Merge toy\_story and taglines on the id column with a **left join**, and save the result as toystory\_tag. Print the rows and type of toystory\_tag.
* With toy\_story as the left table, merge to it taglines on the id column with an **inner join**, and save as toystory\_tag. Priint the rows and type of toystory\_tag.
* **Observe the differences.**

**Right Join**

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.

**A diagram of a right and right data frame

Description automatically generated** **A diagram of a table

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

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.

A screenshot of a computer

Description automatically generated

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.

A screenshot of a computer program

Description automatically generated

**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.

A diagram of a data frame

Description automatically generated A diagram of a table

Description automatically generated

To illustrate 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.

A screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

**Let’s Practice.**

# Right join to find unique movies.

Most of the recent big-budget science fiction movies can also be classified as action movies. You are working with 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 is loaded from movies.p

scifi\_movies, and action\_movies tables are created by subsetting movie\_to\_genres DataFrame (which is loaded from tdmb\_movies\_to\_genres.csv with ‘Science Fiction’ and ‘Action’.

1. **Complete the following tasks.**

* Merge action\_movies and scifi\_movies tables with a right join on movie\_id. Save the result as action\_scifi.
* Update the merge to add suffixes, where '\_act' and '\_sci' are suffixes for the left and right tables, respectively.
* From action\_scifi, subset only the rows where the genre\_act column is null.
* 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.

# 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 are loaded from pop\_movies.csv and tdmb\_movie\_to\_genres.csv, respectively.

1. **Complete the following tasks.**

* 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.
* Plot the bar chart.

**A screenshot of a graph

Description automatically generated**

**Merging a table to itself – Movie Sequel**

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.

sequel table can be loaded from sequels.p

A screenshot of a movie

Description automatically generated

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.

A screenshot of a computer

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

A close-up of a white background

Description automatically generated

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.

A screenshot of a computer program

Description automatically generated

**Let’s Practice.**

# 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 can be loaded from crews.p.

1. **Complete the following tasks.**

* 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.
* Create a Boolean, 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.

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

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

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

**Concatenate DataFrame Together Vertically.**

So far, 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.

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

We can pass a list of table names into pd.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.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**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.

A screenshot of a computer code

Description automatically generated

**Concatenate tables with different columns.**

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 results that the billing country for January invoices is NaN. However, there are values for the February invoices.

A screenshot of a computer screen

Description automatically generatedA blue and white screen with white text

Description automatically generated

If we only want the matching columns between tables, we can 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 only left with the columns the tables have in common.

A white and black rectangular object with numbers

Description automatically generated with medium confidence

**Let’s try.**

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.

Load tracks\_master, tracks\_ride and tracks\_st from tracks\_master.csv, tracks\_ride.csv and tracks\_s.csv, respectively.

1. **Complete the following tasks.**
   * Concatenate tracks\_master, tracks\_ride, and tracks\_st, in that order, setting sort to True.
   * Concatenate tracks\_master, tracks\_ride, and tracks\_st, where the index goes from 0 to n-1.
   * Concatenate tracks\_master, tracks\_ride, and tracks\_st, showing only columns that are in all tables.

# 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.

1. **Complete the following tasks.**
   * 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.

**Merge\_ordered() in Pandas**

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.

A diagram of a table

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A table with green squares

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated

A close up of a sign

Description automatically generated

**Let’s Practice.**

# 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, named sp500, and gdp, can be loaded from S&P500.csv and GDP.csv, respectively.

1. **Complete the following tasks.**
   * Step1.
     1. Use merge\_ordered() to merge gdp and sp500 using a left join on year and date. Save the results as gdp\_sp500.
     2. Print gdp\_sp500 and look at the returns for the year 2018.
   * Step 2
     1. 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.
   * Step 3
     1. Subset the gdp\_sp500 table, select the gdp and returns columns, and save as gdp\_returns.
     2. Print the correlation matrix of the gdp\_returns table using the .corr() method.

# 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 can be loaded from unemployment.csv and inflation.csv, respectively.

1. **Complete the following tasks.**
   * 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.