Exercise1

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

- 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.

In [ ]:
```
# Merge the taxi_owners and taxi_veh tables
taxi_own_veh = taxi_owners.____
# 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)

# Print the value_counts to find the most popular fuel_type
print(taxi_own_veh['----'].value_counts())

#_____________________________#
#Solutions

# Merge the taxi_owners and taxi_veh tables
taxi_own_veh = taxi_owners.merge(taxi_veh,on='vid')
# 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)

# Print the value_counts to find the most popular fuel_type
print(taxi_own_veh['fuel_type'].value_counts())
```

Exercise2

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

- Merge wards and census on the ward column and save the result to wards_census.
- Merge the wards_altered and census tables on the ward column, and notice the difference in returned rows.
- Merge the wards and census_altered tables on the ward column, and notice the difference in returned rows.

In [ ]:
```
# 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)

# 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)

# 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.____
# Print the shape of wards_census_altered
print('wards_census_altered table shape:', wards_census_altered.shape)

#_____#
#Solutions

# 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)
```

```
# 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)



# 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)
```

Exercise3

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

- 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.

In [ ]:
```
# 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(____)

#_____________________________#
#Solutions

# 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())
```

Exercise4

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

- 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.
- 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.

In [ ]:
```
# Merge the ridership and cal tables
ridership_cal = ridership.merge(____)

# Merge the ridership, cal, and stations tables
ridership_cal_stations = ridership.merge(cal, on=['year','month','day']) \
                            .merge(____)

# 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'] == ____)
                   & (ridership_cal_stations['day_type'] == ____)
                   & (ridership_cal_stations['station_name'] == ____))

# Use .loc and the filter to select for rides
print(ridership_cal_stations.loc[filter_criteria, 'rides'].sum())

#_____________________________#
#Solutions

# Merge the ridership and cal tables
ridership_cal = ridership.merge(cal)

# Merge the ridership, cal, and stations tables
ridership_cal_stations = ridership.merge(cal, on=['year','month','day']) \
                            .merge(stations, on='station_id')

# 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())
```

Exercise5

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

- 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.

In [ ]:
```
# 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'}))

#_____________________________#
#Solutions

# 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'}))
```

Exercise6

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

- 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.
- 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.
- Sort pop_vac_lic by vacant, account, andpop_2010 in descending, ascending, and ascending order respectively. Save it as sorted_pop_vac_lic.

In [ ]:
```
# Merge land_use and census and merge result with licenses including suffixes
land_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(____,
                                   as_index=False).agg({'account':'count'})

#_____________________________#
#Solutions

# 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'))

# 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=____)


# 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())
```