Exercise1

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

- 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
- Count the number of rows in movies_financials with a null value in the budget column.

In [ ]:

```
# Merge movies and financials with a left join
movies_financials = movies.merge(____)

# 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)

#_____________________#
#Solutions

# Merge movies and financials with a left join
movies_financials = movies.merge(financials,how='left')

# 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'].isna().sum()

# Print the number of movies missing financials
print(number_of_missing_fin)
```

Exercise2

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

- Merge toy_story and taglines on the id column with a left join, and save the result as 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.

In [ ]:

```
# Merge the toy_story and taglines tables with a left join
toystory_tag = toy_story.merge(____)

# Merge the toy_story and taglines tables with a inner join
toystory_tag = ____

# Print the rows and shape of toystory_tag
print(toystory_tag)
print(toystory_tag.shape)

#_____________________#
#Solutions

# 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 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)
```

Exercise3

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

- 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.

In [ ]:

```
# Merge action_movies to scifi_movies with right join
action_scifi = ____.merge(____)

# 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())

# From action_scifi, select only the rows where the genre_act column is null
scifi_only = action_scifi[____]

# 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'))

# 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)

#_____________________#
#Solutions

# Merge action_movies to scifi_movies with right join
action_scifi = action_movies.merge(scifi_movies,on='movie_id',how='right')

# 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())

# From action_scifi, select only the rows where the genre_act column is null
scifi_only = action_scifi[action_scifi['genre_act'].isnull()]

# 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'))

# 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)
```

Exercise4

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

- 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.

In [ ]:

```
# 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()

#____________________#
#Solutions

# 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()
```

Exercise5

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

- 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.

In [ ]:

```
# 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())

#____________________#
#Solutions

# 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())
```

Exercise6

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

- 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 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.
- Use the .head() method to print the first few rows of direct_crews.

In [ ]:

```
# Merge the crews table to itself
crews_self_merged = ____

# 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]

# Print the first few rows of direct_crews

#____________________#
#Solutions

# Merge the crews table to itself
crews_self_merged = crews.merge(crews,on='id',suffixes=('_dir','_crew'))

# 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 the first few rows of direct_crews
print(direct_crews.head())
```

Exercise7

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.

In [ ]:

```
# 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())

#_____________________#
#Solutions

# Merge to the movies table the ratings table on the index
movies_ratings = movies.merge(ratings, on='id', how='left')

# Print the first few rows of movies_ratings
print(movies_ratings.head())
```

Exercise8

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

- 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 the sequels_fin table to itself with an inner join, where the left and right tables merge on sequel and id respectively with suffixes equal to ('_org','_seq'), saving to orig_seq.
- Select the title_org, title_seq, and diff columns of orig_seq and save this as titles_diff.
- Sort by titles_diff by diff in descending order and print the first few rows.

In [ ]:

```
# Merge sequels and financials on index id
sequels_fin = ____

# 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']

# Select the title_org, title_seq, and diff
titles_diff = orig_seq[____]

# Print the first rows of the sorted titles_diff
print(titles_diff.sort_values(____).head())

#_____________________#
#Solutions

# 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())
```