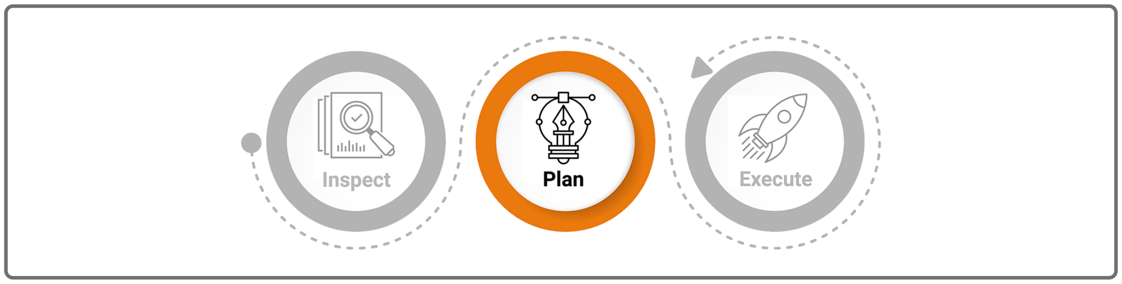
8.4.2 Transform and Merge Rating Data

Britta wants to include the rating data with the movie data, but it's a very large dataset. She wants to reduce the ratings data to a useful summary of rating information for each movie, and then make the full dataset available to the hackathon participants if they decide they need more granular rating information.



For each movie, Britta wants to include the rating data, but the rating dataset has so much information that it's too unwieldy to use all of it. We could calculate some basic statistics like the mean and median rating for each movie, but a more useful summary is just to count how many times a movie received a given rating. This way, someone who wants to calculate statistics for the dataset would have all the information they need.

We'll include the raw ratings data if the hackathon participants want to do more in-depth analysis, such as comparing across users, but having the rating counts for each movie is easy enough to do. Plus, it will enable the hackathon participants to calculate statistics on their own without having to work with a dataset containing 26-million rows.

First, we need to use a `groupby` on the "movieId" and "rating" columns and take the count for each group.

```
rating_counts = ratings.groupby(['movieId','rating'], as_index=False).count(
```

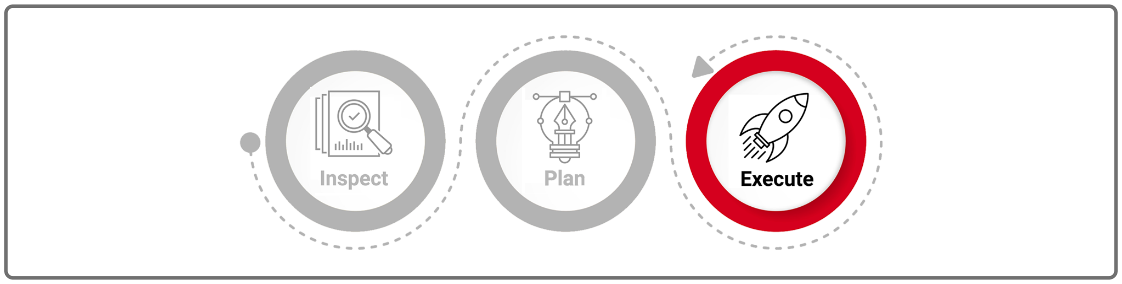
Then we'll rename the "userId" column to "count."

NOTE

> The choice of renaming "userId" to "count" is arbitrary. Both "userId" and "timestamp" have the same information, so we could use either one.

Your code should look like the following:

```
rating_counts = ratings.groupby(['movieId','rating'], as_index=False).count(
                .rename({'userId':'count'}, axis=1)
```



Now the magical part. We can pivot this data so that `movieId` is the index, the columns will be all the rating values, and the rows will be the counts for each rating value.

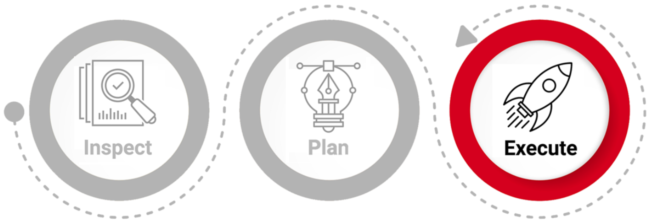
```
rating_counts = ratings.groupby(['movieId','rating'], as_index=False).count(
                .rename({'userId':'count'}, axis=1) \
                .pivot(index='movieId',columns='rating', values='count')
```

We want to rename the columns so they're easier to understand. We'll prepend `rating_` to each column with a list comprehension:

```
rating_counts.columns = ['rating_' + str(col) for col in rating_counts.colum
```

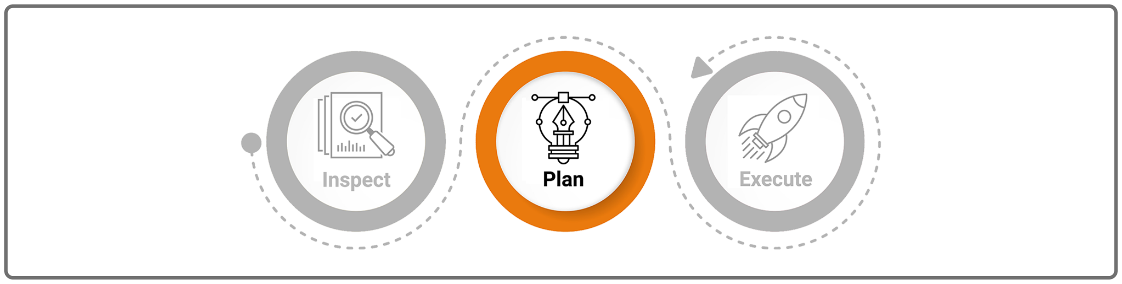
Now we can merge the rating counts into `movies_df`.

Retake



This time, we need to use a left merge, since we want to keep everything in `movies_df`:

```
movies_with_ratings_df = pd.merge(movies_df, rating_counts, left_on='kaggle_
```



Finally, because not every movie got a rating for each rating level, there will be missing values instead of zeros. We have to fill those in ourselves, like this:

```
movies_with_ratings_df[rating_counts.columns] = movies_with_ratings_df[ratin
```

And we're done—we just finished the Transform step in ETL! Now all that's left is loading our tables into SQL.

ADD/COMMIT/PUSH

Remember to add, commit, and push your work!