Pandas DataFrame Operations - Grouping and Sorting

6. Grouping

Things start looking really interesting when we group rows with certain criteria and then aggregate their data.

Say we want to group our dataset by director and see **how much revenue (sum) each director earned at the box-office and then also look at the average rating (mean) for each director**. We can do this by using the `groupby` operation on the column of interest, followed by the appropriate aggregate (*sum/mean*), like so:

```
# Let's group our dataset by director and see how much revenue each director has
movies_df.groupby('Director').sum()

# Let's group our dataset by director and see the average rating of each director
movies_df.groupby('Director')[['Rating']].mean()
```

As we can see, Pandas grouped all the *'Director'* rows by name into one. And since we used *sum()* for aggregation, it added together all the numerical columns. The values for each of the columns now represent the sum of values in that column for that director.

For example, we can see that the director Aamir Khan has a very high average rating (8.5) but his revenue is much lower compared to many other directors (only 1.20M $). This can be attributed to the fact that we are looking at a dataset with international movies, and Hollywood directors/movies have understandably much higher revenues compared to movies from international directors.

In addition to `sum()` and `mean()` Pandas provides multiple other aggregation functions like `min()` and `max()`.

Alt! Can you find a problem in the code when we apply aggregation to get the sum?

This is not the correct approach for all the columns. We do not want to sum all the *'Year'* values, for instance. To make Pandas apply the aggregation on some of the columns only, we can specify the name of the columns we are interested in. For example, in the second example, we specifically passed the *'Rating'* column, so that the mean did not get applied to all the columns.

Grouping is an easy and extremely powerful data analysis method. We can use it to fold datasets and uncover insights from them, while aggregation is one of the foundational tools of statistics. In fact, *learning to use `groupby()` to its full potential can be one of the greatest uses of the Pandas library*.

7. Sorting

Pandas allows easy sorting based on multiple columns. We can apply sorting on the result of the *groupby()* operation or we can apply it directly to the full DataFrame. Let's see this in action via two examples:

1. Say we want the **total revenue per director and to have our results sorted by earnings**, not in alphabetical order like in the previous examples.

We can first do a *groupby()* followed by *sum()* (just like before) and then we can call `sort_values` on the results. To sort by revenue, we need to pass the name of that column as input to the sorting method; we can also specify that we want results sorted from highest to lowest revenue:

```
#Let's group our dataset by director and see who earned the most
movies_df.groupby('Director')[['Revenue (Millions)']].sum().sort_values(['Revenue (Millions)'
```



```
#Let's group our dataset by director and see who earned the most
movies_df.groupby('Director')[['Revenue (Millions)']].sum().sort_values(['Revenue (Millions)'], ascending=False)
```

Director	Revenue (Millions)
J.J. Abrams	1683.45
David Yates	1830.51
Christopher Nolan	1516.09
Michael Bay	1421.32
Francis Lawrence	1299.91
Joss Whedon	1050.27
Jon Favreau	1025.90
Zack Snyder	975.74
Peter Jackson	900.45

2. Now, say we want to see **which movies had both the highest revenue and the highest rating**:

Any guesses?!

```
# Let's sort our movies by revenue and ratings and then get the top 10 results
data_sorted = movies_df_title_indexed.sort_values(['Revenue (Millions)','Rating'], ascending=
data_sorted[['Revenue (Millions)','Rating']].head(10)
```

```
#Pandas allow easy sorting based on multiple columns as in this example
#Let's sort our movies by revenue and ratings and then get the top 10 results
data_sorted = movies_df_indexed.sort_values(['Revenue (Millions)','Rating'], ascending=False)
data_sorted[['Revenue (Millions)','Rating']].head(10)
```

Title	Revenue (Millions)	Rating
Star Wars: Episode VII - The Force Awakens	936.63	8.1
Avatar	760.51	7.8
Jurassic World	652.18	7.0
The Avengers	623.28	8.1
The Dark Knight	533.32	9.0
Rogue One	532.17	7.9
Finding Dory	486.29	7.4
Avengers: Age of Ultron	458.99	7.4
The Dark Knight Rises	448.13	8.5
The Hunger Games: Catching Fire	424.65	7.6

We now know that **J.J. Abrams is the director who earned the most $$$ at the box office, and that Star Wars is the movie with the highest revenue and rating**, followed by some other very popular movies!

Star Wars: The Force Awakens (Image Source: starwars.com)

Jupyter Notebook

You can see the instructions running in the Jupyter Notebook below:

📝 **How to Use a Jupyter NoteBook?**

- Click on **"Click to Launch"** 🚀 button to work and see the code running live in the notebook.

- You can click ⬈ to open the **Jupyter Notebook in a new tab**.

- Go to files and click *Download as* and then choose the format of the file to **download** ⬇. You can choose Notebook(.ipynb) to download the file and work locally or on your personal Jupyter Notebook.

- ⚠ The notebook **session expires after 30 minutes of inactivity**. It will reset if there is no interaction with the notebook for 30 consecutive minutes.

Your app can be found at: https://1dgnrwwoynk5m-live-app.educative.run/notebooks/Grouping%26Sorting.ipynb ⬈

Click to launch app!