Pandas Groupby

(https://github.com/learn-co-curriculum/dsc-pandas-groupby) (https://github.com/learn-co-curriculum/dsc-pandas-groupby/issues/new/choose)

Introduction

In this lab, you'll learn how to use the `.groupby()` method in Pandas to summarize datasets.

Objectives

You will be able to:

- Use groupby methods to aggregate different groups in a dataframe

Using `.groupby()`

Consider an example of the titanic DataFrame:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

During the Exploratory Data Analysis phase, one of the most common tasks you'll want to do is split the dataset into subgroups and compare them to see if you can notice any trends. For instance, you may want to group the passengers together by gender or age. You can do this by using the `.groupby()` method built-in to pandas DataFrames.

To group passengers by gender, you would type:

```
df.groupby('Sex')
```

```
df.groupby(df['Sex'])
```

Note that this alone will not display a result -- although you have split the dataset into groups, you don't have a meaningful way to display information until you chain an ***Aggregation Function*** onto the groupby. This allows you to compute summary statistics!

You can quickly use an aggregation function by chaining the call to the end of the `.groupby()`
m

Help

```
df.groupby('Sex').sum()
```

The code above returns displays the following DataFrame:

Sex	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
female	135343	233	678	7286.00	218	204	13966.6628
male	262043	109	1379	13919.17	248	136	14727.2865

You can use aggregation functions to quickly help us compare subsets of our data. For example, the aggregate statistics displayed above allow you to quickly notice that there were more female survivors overall than male survivors.

Aggregation functions

There are many built-in aggregate methods provided for you in the `pandas` package, and you can even write and apply your own. Some of the most common aggregate methods you may want to use are:

- `.min()` : returns the minimum value for each column by group
- `.max()` : returns the maximum value for each column by group
- `.mean()` : returns the average value for each column by group
- `.median()` : returns the median value for each column by group
- `.count()` : returns the count of each column by group

You can also see a list of all of the built-in aggregation methods by creating a grouped object and then using tab completion to inspect the available methods:

```
grouped_df = df.groupby('Sex')
grouped_df.<TAB>
```

This will display the following output:

```
In [26]: grouped_df.<TAB>
gb.agg         gb.boxplot     gb.cummin      gb.describe    gb.filter      gb.get_group   gb.he
gb.aggregate   gb.count       gb.cumprod     gb.dtype       gb.first       gb.groups      gb.hi
gb.apply       gb.cummax      gb.cumsum      gb.fillna      gb.gender      gb.head        gb.in
```

This is a comprehensive list of all built-in methods available to grouped objects. Note that some are aggregation methods, while others, such as `gb.fillna()`, allow us to fill missing values to individual groups independently.

Help

Multiple groups

You can also split data into multiple different levels of groups by passing in an array containing the name of every column you want to group by -- for instance, by every combination of both `Sex` and `Pclass`.

```
df.groupby(['Sex', 'Pclass']).mean()
```

The code above would return the following DataFrame:

		PassengerId	Survived	Age	SibSp	Parch	Fare
Sex	**Pclass**						
female	**1**	469.212766	0.968085	34.611765	0.553191	0.457447	106.125798
	2	443.105263	0.921053	28.722973	0.486842	0.605263	21.970121
	3	399.729167	0.500000	21.750000	0.895833	0.798611	16.118810
male	**1**	455.729508	0.368852	41.281386	0.311475	0.278689	67.226127
	2	447.962963	0.157407	30.740707	0.342593	0.222222	19.741782
	3	455.515850	0.135447	26.507589	0.498559	0.224784	12.661633

Selecting information from grouped objects

Since the resulting object returned is a DataFrame, you can also slice a selection of columns you're interested in from the DataFrame returned.

The example below demonstrates the syntax for returning the mean of the `Survived` class for every combination of `Sex` and `Pclass`:

```
df.groupby(['Sex', 'Pclass'])['Survived'].mean()
```

The code above returns the following DataFrame:

```
Sex     Pclass
female  1         0.968085
        2         0.921053
        3         0.500000
male    1         0.368852
        2         0.157407
        3         0.135447
Name: Survived, dtype: float64
```

The above example slices by column, but you can also slice by index. Take a look:

```
grouped = df.groupby(['Sex', 'Pclass'])['Survived'].mean()
grouped['female']
```

Help

```
print(grouped['female'][1])
```

Note that you need to provide only the value `female` as the index, and are returned all the groups where the passenger is female, regardless of the `Pclass` value. The second example shows the results for female passengers with a 1st-class ticket.

Summary

In this lab, you learned about how to split a DataFrame into subgroups using the `.groupby()` method. You also learned to generate aggregate views of these groups by applying built-in methods to a groupby object.

How do you feel about this lesson?

Have specific feedback?

Tell us here! (https://github.com/learn-co-curriculum/dsc-pandas-groupby/issues/new/choose)

Help