Pandas Groupby — Explained

How to efficiently use Groupby function of Pandas

Pandas is a very powerful Python data analysis library that expedites the preprocessing steps of your project. In this post, I will cover **groupby** function of Pandas with many examples that help you gain a comprehensive understanding of the function.

Photo by Markus Spiske on Unsplash

Groupby is a versatile and easy-to-use function that helps to get an overview of the data. It makes it easier to explore the dataset and unveil the underlying relationships among variables.

Groupby is best explained over examples. I will use a customer churn dataset available on Kaggle. I only took a part of it which is enough to show every detail of groupby function.

As always, we start with importing NumPy and pandas:

```
import pandas as pd
import numpy as np
```

Let’s take a quick look at the dataset:

```
df.shape
(7043, 9)df.head()
```

The dataset includes 8 features about a customer and whether or not a customer churned (i.e. left the company).

Before applying groupby function to the dataset, let’s go over a visual example. Visualizations are always a good way to explain concepts. Assume we have two features. One is color which is a categorical feature and the other one is a numerical feature, values. We want to **group** values **by** color and calculate the **mean** (or any other aggregation) of values for different colors. Then finally **sort** the colors based on average values. The following figure shows the steps of this process.

Let’s go back to our dataset. To see if gender plays a role in churning, we can take the **gender** and **churn** columns and group them by **gender**. Then we can apply the mean function.

Churn rates for males and females are very close.

If we do not select columns and directly apply groupby to the dataframe, all numerical columns will be calculated according to the aggregation function:

It seems like all numerical features are approximately the same on average for males and females.

We can also sort the result by adding the **sort_values** function:

It sorts in ascending order by default. We can change it by setting the **ascending** parameter as false.

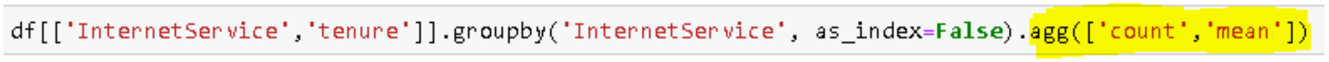
Customers who have fiber optic internet service are much more likely to churn than other customers.

We can group by based on multiple columns by passing the column names in square brackets:

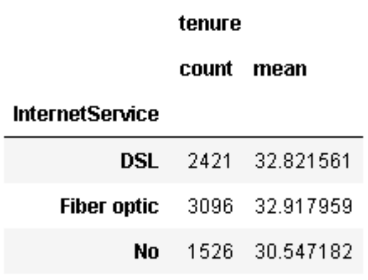
“Contract” column has 3 categories and “SeniorCitizen” column has 2 categories so we have a total of 6 groups. For each group, average “Churn” rate is calculated.

Multiple aggregate functions

In some cases, after you applied groupby function, you may want to see both the count and mean of different groups. If the class distribution is not balanced, only checking the mean may cause false assumptions. You can apply multiple aggregate functions on the result of groupby. They are not limited to the only count and mean, you can pass the name of the functions as an argument to **agg()** function.



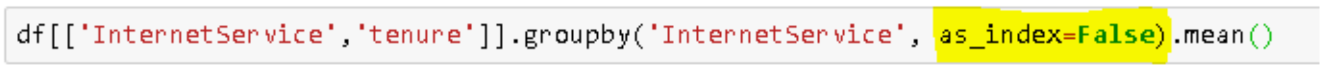
```
df[['InternetService','tenure']].groupby('InternetService', as_index=False).agg(['count','mean'])
```



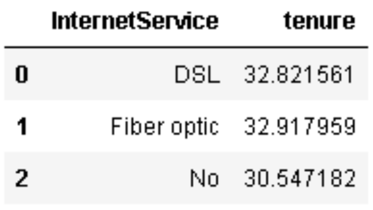
	tenure	
	count	mean
InternetService		
DSL	2421	32.821561
Fiber optic	3096	32.917959
No	1526	30.547182

as_index parameter

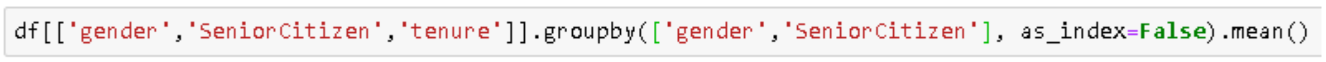
The variables in the groupby function are returned as the index of the resulting dataframe. We can change it by setting **as_index** parameter as false.



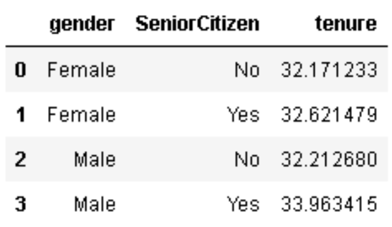
```
df[['InternetService','tenure']].groupby('InternetService', as_index=False).mean()
```



	InternetService	tenure
0	DSL	32.821561
1	Fiber optic	32.917959
2	No	30.547182

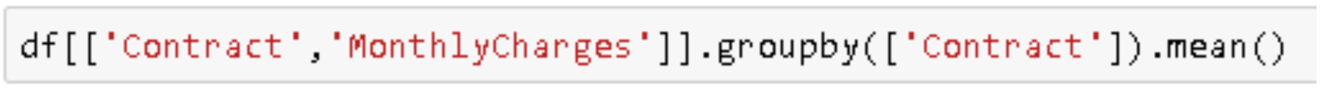


```
df[['gender','SeniorCitizen','tenure']].groupby(['gender','SeniorCitizen'], as_index=False).mean()
```

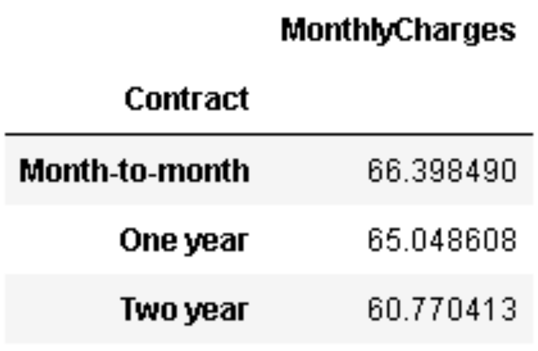


	gender	SeniorCitizen	tenure
0	Female	No	32.171233
1	Female	Yes	32.621479
2	Male	No	32.212680
3	Male	Yes	33.963415

Let’s find out if monthly charges depend on the contract type. We need to take the “contract” and “MonthlyCharges” columns from the dataset, groupby “contract” and apply mean function on the result.



```
df[['Contract','MonthlyCharges']].groupby(['Contract']).mean()
```



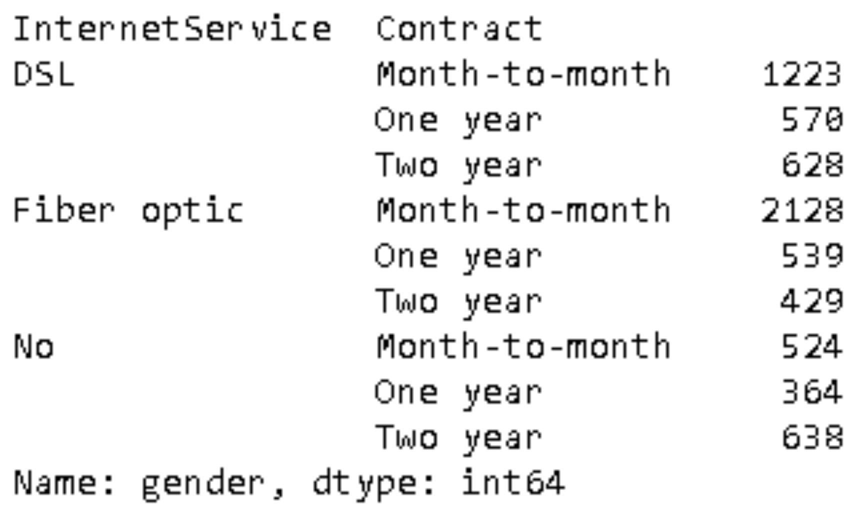
	MonthlyCharges
Contract	
Month-to-month	66.398490
One year	65.048608
Two year	60.770413

As expected, long-term contracts have lower monthly charges on average.

We can apply other aggeragate functions depending on the task. For example, we may want to see the distribution of contract type for different internet service types. Since both are categorical variables, we can apply count function:

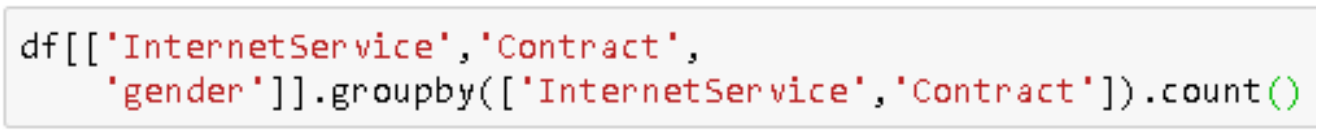
Since we applied count function, the returned dataframe includes all other columns because it can count the values regardless of the dataframe. The number of values is the same on all the columns, so we can just select one column to see the values. One of the nice things about Pandas is that there is usually more than one way to accomplish a task.

These three lines return the following output:

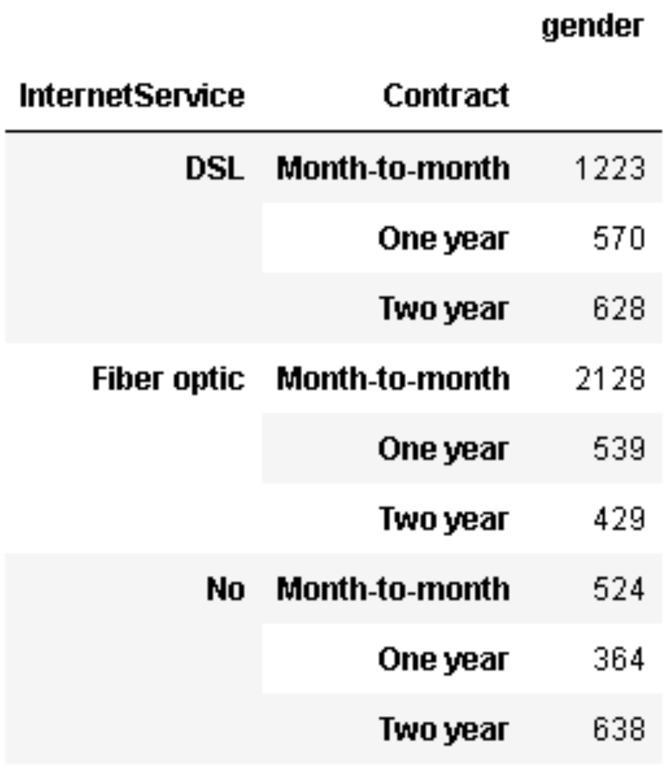


```
InternetService  Contract
DSL              Month-to-month    1223
                 One year           570
                 Two year           628
Fiber optic      Month-to-month    2128
                 One year           539
                 Two year           429
No               Month-to-month     524
                 One year           364
                 Two year           638
Name: gender, dtype: int64
```

You can also do the same task as follows but the returned object will be a dataframe:



```
df[['InternetService','Contract',
    'gender']].groupby(['InternetService','Contract']).count()
```



		gender
InternetService	**Contract**	
DSL	**Month-to-month**	1223
	One year	570
	Two year	628
Fiber optic	**Month-to-month**	2128
	One year	539
	Two year	429
No	**Month-to-month**	524
	One year	364
	Two year	638

Please note that some aggregate functions require numerical values. For example, if we only have categorical variables and try to apply mean function, we will get an error:

We can diversify the examples but the underlying logic is the same. Groupby function can be used as the first step in exploratory data analysis process because it gives us an idea about the relationship between variables in the dataset.

Thanks for reading. Please let me know if you have any feedback.