

✓ Groupby

There are certain situations where you need to find groups in columns of a dataframe. You might want to perform specific actions to specific groups. For example, let's say you have a database of all the students at a university. You want to group all the students by "current" or "alumni." The database has a column called "Status." For every student, it has either a value of "current" or a value of "alumni." These kinds of features that separate the data into groups, or categories, are called categorical features. The command groupby is used predominantly for categorical grouping variables.

As you move through the examples, you can understand better. Let's look at the following dataframe.

```
import numpy as np
dic = {
    'Name': ['Bob', 'Sam', 'Mary', 'Alice', 'Beth'],
    'Status': ['Current', 'Alumni', 'Alumni', 'Alumni', 'Current'],
    'Fees': ['Due', 'Clear', 'Scholarship', 'Scholarship', 'Due'],
    'Score': [78, 68, 89, 87, 76],
    'Score2': [67, 87, 96, 95, np.nan]
}
import pandas as pd
df = pd.DataFrame(dic)
df
```

Categorical Data: Certain sets of unique values



	Name	Status	Fees	Score	Score2
0	Bob	Current	Due	78	67.0
1	Sam	Alumni	Clear	68	87.0
2	Mary	Alumni	Scholarship	89	96.0
3	Alice	Alumni	Scholarship	87	95.0
4	Beth	Current	Due	76	NaN


As you can see, there are two categorical features. One is status, and the other is fees. As a side note, notice that one of the values for the score is missing.

Now, let's group the students by status.

```
status = df.groupby('Status')
```

Now let us see how many groups there are.

```
status.groups
```




```
{'Alumni': [1, 2, 3], 'Current': [0, 4]}
```

We can see the two groups. If you can imagine a large dataset with many groups, it might be beneficial to be able to group them and see how many there are. The above output above also contains other information about how the computer is storing these groups in memory.

Just for good measure, let's make another group for the "Fees" column.


```
df
```

```
# "Group those students by "Status" and only print out the current students"
```




	Name	Status	Fees	Score	Score2
0	Bob	Current	Due	78	67.0
1	Sam	Alumni	Clear	68	87.0
2	Mary	Alumni	Scholarship	89	96.0
3	Alice	Alumni	Scholarship	87	95.0
4	Beth	Current	Due	76	NaN

```
fees = df.groupby('Fees')
fees.groups
```




```
{'Clear': [1], 'Due': [0, 4], 'Scholarship': [2, 3]}
```

```
for fee_status, students in fees.groups.items():
    if fee_status=="Scholarship":
        for student in students:
            print(df.iloc[student])
```




```
Name      Mary
Status    Alumni
Fees      Scholarship
Score      89
Score2     96
Name: 2, dtype: object
Name      Alice
Status    Alumni
Fees      Scholarship
Score      87
Score2     95
Name: 3, dtype: object
```

```
df
```



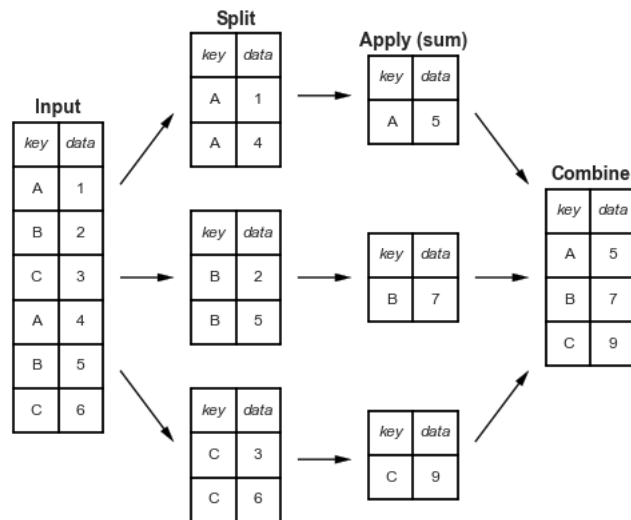
	Name	Status	Fees	Score	Score2
0	Bob	Current	Due	78	67.0
1	Sam	Alumni	Clear	68	87.0
2	Mary	Alumni	Scholarship	89	96.0
3	Alice	Alumni	Scholarship	87	95.0
4	Beth	Current	Due	76	NaN

```
df.groupby("Status").max()
```



	Name	Fees	Score	Score2
Status				
Alumni	Sam	Scholarship	89	96.0
Current	Bob	Due	78	67.0

Internal operations in GroupBy (Split, apply, combine)



Source Image: <https://jakevdp.github.io/PythonDataScienceHandbook/03.08-aggregation-and-grouping.html>

The GroupBy operation:

1. First, it splits the dataframe based on the specified key.
2. Second, it aggregates, transforms, or filter the individual groups in the application step.
3. Finally, it merges the results of the operations in the combining step.

We can use other operations of Pandas to perform this operation; the groupBy operation performs this in a single pass, i.e., it abstracts away the manual steps.

✓ Performing actions on groups

Now that we have successfully grouped items, we can perform computations on these different groups.

Counting

We can count how many students there are in each group.

```
status.count()
```



	Name	Fees	Score	Score2
Status				
Alumni	3	3	3	3
Current	2	2	2	1

There are 3 entries for all columns of the group alumni. There are two current students, but one of the scores is missing. Hence, we get the output 1.

✓ sum

We can sum the elements of the group.

```
status.sum()
```



	Score	Score2
Status		
Alumni	244	278.0
Current	154	67.0

▼ max

We can also group by maximum value.

```
print("Max")
print(status.max())
```

```
↗ Max
```

	Name	Fees	Score	Score2
Status				
Alumni	Sam	Scholarship	89	96.0
Current	Bob	Due	78	67.0

Start coding or [generate](#) with AI.

Similarly, we can group by minimum value.

▼ Finding unique values

In some cases, we might want to find the number of unique values in a dataset. For example, you might want to know about the home countries of the students. This function only counts unique values, so even if a country is repeated, then it counts the country once.

```
status.nunique()
```

```
↗
```

	Name	Status	Fees	Score	Score2
Status					
Alumni	3	1	2	3	3
Current	2	1	1	2	1

There are five different names, two different statuses, three different fees, and so on.

You can also numerically compare the different groups we have made using the groupby method in pandas. For example, one might imagine a scenario where they want to find the mean of the scores of the current students, in comparison to older ones. The following command can perform this calculation.

```
status.mean()
```

```
↗
```

	Score	Score2
Status		
Alumni	81.333333	92.666667
Current	77.000000	67.000000

Notice that pandas automatically discards all categorical features, since they have strings. It also throws away any missing values. It looks like the Alumni have higher marks than the current students.

▼ Grouping more than one column

Grouping more than one column

You might want to analyze two columns at once. You might be trying to find how many alumni still have their fees pending. You can do so using the following commands.

```
new_group = df.groupby(['Status', 'Fees'])
new_group.size()
```

```
↗
```

Status	Fees	
Alumni	Clear	1
	Scholarship	2
Current	Due	2

dtype: int64

According to the output above, there were two alumni on scholarship, and one has their dues cleared. Two current students have their fees due.

Notice that we used the `.size()` method. It is similar to `count()` . It just gives back the size of the group.

Aggregations

We have learned that we can perform many aggregations in Pandas. With `aggregate()`, you can even perform more than one aggregates at once.

Let us suppose you are analyzing data from a business. You might want to perform many statistical tests on the data. Let us apply different statistical measures on the data.

```
import numpy as np
dic = {
    'Name': ['Bob', 'Sam', 'Mary', 'Alice', 'Beth'],
    'Status': ['Current', 'Alumni', 'Alumni', 'Alumni', 'Current'],
    'Fees': ['Due', 'Clear', 'Scholarship', 'Scholarship', 'Due'],
    'Score': [78, 68, 89, 87, 76],
    'Score2': [67, 87, 96, 95, np.nan],
    'Score3': [100, 200, 300, 400, 500]
}
import pandas as pd
df = pd.DataFrame(dic)
df
```

	Name	Status	Fees	Score	Score2	Score3
0	Bob	Current	Due	78	67.0	100
1	Sam	Alumni	Clear	68	87.0	200
2	Mary	Alumni	Scholarship	89	96.0	300
3	Alice	Alumni	Scholarship	87	95.0	400
4	Beth	Current	Due	76	NaN	500

```
status = df.groupby("Status")
```

```
status.agg(['mean', 'median', 'std', 'min', 'max'])
```

	Score			Score2			Score3								
	mean	median	std	min	max	mean	median	std	min	max	mean	median	std	min	max
Status															
Alumni	81.333333	87	11.590226	68	89	92.666667	95.0	4.932883	87.0	96.0	300	300	100.000000	200	400
Current	77.000000	77	1.414214	76	78	67.000000	67.0	NaN	67.0	67.0	300	300	282.842712	100	500

The code above used the `agg()` method. This method takes as input a list of strings. You can type in the required statistical measure and get results.

References

Documentation