Pandas: 5th lesson – Grouping & Sorting

Group-wise analysis:

* One function we've been using heavily thus far is the value\_counts() function. We can replicate what value\_counts() does by doing the following:

import pandas as pd

reviews = pd.read\_csv("../input/wine-reviews/winemag-data-130k-v2.csv", index\_col=0)

pd.set\_option("display.max\_rows", 5)

reviews.groupby('points').points.count()

points

80 397

81 692

...

99 33

100 19

Name: points, Length: 21, dtype: int64

* groupby() created a group of reviews which allotted the same point values to the given wines. Then, for each of these groups, we grabbed the points() column and counted how many times it appeared. value\_counts() is just a shortcut to this groupby() operation. We can use any of the summary functions we've used before with this data. For example, to get the cheapest wine in each point value category, we can do the following:

reviews.groupby('points').price.min()

points

80 5.0

81 5.0

...

99 44.0

100 80.0

Name: price, Length: 21, dtype: float64

* You can think of each group we generate as being a slice of our DataFrame containing only data with values that match. This DataFrame is accessible to us directly using the apply() method, and we can then manipulate the data in any way we see fit. For example, here's one way of selecting the name of the first wine reviewed from each winery in the dataset:

reviews.groupby('winery').apply(lambda df: df.title.iloc[0])

winery

1+1=3 1+1=3 NV Rosé Sparkling (Cava)

10 Knots 10 Knots 2010 Viognier (Paso Robles)

...

àMaurice àMaurice 2013 Fred Estate Syrah (Walla Walla V...

Štoka Štoka 2009 Izbrani Teran (Kras)

Length: 16757, dtype: object

* For even more fine-grained control, you can also group by more than one column. For an example, here's how we would pick out the best wine by country and province:

reviews.groupby(['country', 'province']).apply(lambda df: df.loc[df.points.idxmax()])

* Another groupby() method worth mentioning is agg(), which lets you run a bunch of different functions on your DataFrame simultaneously. For example, we can generate a simple statistical summary of the dataset as follows:

reviews.groupby(['country']).price.agg([len, min, max])

len min max

country

Argentina 3800 4.0 230.0

Armenia 2 14.0 15.0

... ... ... ...

Ukraine 14 6.0 13.0

Uruguay 109 10.0 130.0

43 rows × 3 columns

Effective use of groupby() will allow you to do lots of really powerful things with your dataset.

Multi-indexes:

* In all of the examples we've seen thus far we've been working with DataFrame or Series objects with a single-label index. groupby() is slightly different in the fact that, depending on the operation we run, it will sometimes result in what is called a multi-index. A multi-index differs from a regular index in that it has multiple levels. For example:

countries\_reviewed = reviews.groupby(['country', 'province']).description.agg([len])

countries\_reviewed

len

country province

Argentina Mendoza Province 3264

Other 536

... ... ...

Uruguay San Jose 3

Uruguay 24

425 rows × 1 columns

mi = countries\_reviewed.index

type(mi)

pandas.core.indexes.multi.MultiIndex

* Multi-indices have several methods for dealing with their tiered structure which are absent for single-level indices. They also require two levels of labels to retrieve a value. Dealing with multi-index output is a common "gotcha" for users new to pandas. The use cases for a multi-index are detailed alongside instructions on using them in the MultiIndex / Advanced Selection section of the pandas documentation. However, in general the multi-index method you will use most often is the one for converting back to a regular index, the reset\_index() method:

countries\_reviewed.reset\_index()

country province len

0 Argentina Mendoza Province 3264

1 Argentina Other 536

... ... ... ...

423 Uruguay San Jose 3

424 Uruguay Uruguay 24

425 rows × 3 columns

Sorting:

* Looking again at countries\_reviewed, we can see that grouping returns data in index order, not in value order. That is to say, when outputting the result of a groupby, the order of the rows is dependent on the values in the index, not in the data. To get data in the order want it in we can sort it ourselves. The sort\_values() method is handy for this:

countries\_reviewed = countries\_reviewed.reset\_index()

countries\_reviewed.sort\_values(by='len')

country province len

179 Greece Muscat of Kefallonian 1

192 Greece Sterea Ellada 1

... ... ... ...

415 US Washington 8639

392 US California 36247

425 rows × 3 columns

* sort\_values() defaults to an ascending sort, where the lowest values go first. However, most of the time we want a descending sort, where the higher numbers go first. That goes thusly:

countries\_reviewed.sort\_values(by='len', ascending=False)

country province len

392 US California 36247

415 US Washington 8639

... ... ... ...

63 Chile Coelemu 1

149 Greece Beotia 1

425 rows × 3 columns

* To sort by index values, use the companion method sort\_index(). This method has the same arguments and default order:

countries\_reviewed.sort\_index()

country province len

0 Argentina Mendoza Province 3264

1 Argentina Other 536

... ... ... ...

423 Uruguay San Jose 3

424 Uruguay Uruguay 24

425 rows × 3 columns

* Finally, know that you can sort by more than one column at a time:

countries\_reviewed.sort\_values(by=['country', 'len'])

country province len

1 Argentina Other 536

0 Argentina Mendoza Province 3264

... ... ... ...

424 Uruguay Uruguay 24

419 Uruguay Canelones 43

425 rows × 3 columns