tut_3_b

March 11, 2025

1 Introduction

Maps allow us to transform data in a DataFrame or Series one value at a time for an entire column. However, often we want to group our data, and then do something specific to the group the data is in.

As you'll learn, we do this with the `groupby()` operation. We'll also cover some additional topics, such as more complex ways to index your DataFrames, along with how to sort your data.

2 Understanding Groupby¶

ฟังก์ชัน groupby() ใน Pandas จะแบ่งข้อมูลทั้งหมดจากชุดข้อมูลออกเป็นหมวดหมู่หรือกลุ่มต่างๆ ทำให้สามารถวิเคราะห์ข้อมูล ตามกลุ่มต่างๆ ได้อย่างยืดหยุ่น

Here's a super simple dataframe to illustrate some examples. We'll be grouping the data by the "animal" column where there are four categories of animals:

- alligators
- cats
- snakes
- hamsters

[ ]:
```
import numpy as np
import pandas as pd
import random

# Random pets column
pet_list = ["cat", "hamster", "alligator", "snake"]
pet = [random.choice(pet_list) for i in range(1,15)]

# Random weight of animal column
weight = [random.choice(range(5,15)) for i in range(1,15)]
```

```
# Random length of animals column
length = [random.choice(range(1,10)) for i in range(1,15)]

# random age of the animals column
age = [random.choice(range(1,15)) for i in range(1,15)]

# Put everyhting into a dataframe
df = pd.DataFrame()
df["animal"] = pet
df["age"] = age
df["weight"] = weight
df["length"] = length

# make a groupby object
animal_groups = df.groupby("animal")
```

[ ]:
```
df
```

- เราสามารถถามเกี่ยวกับข้อมูลสัตว์ได้ก็คือ
- หากต้องการหาค่าเฉลี่ย(**mean**)ของน้ำหนักของสัตว์แต่ละประเภท เราจะจัดกลุ่มสัตว์ตามประเภทของสัตว์ จากนั้นจึงใช้ฟังก์ชันค่าเฉลี่ย
- เราสามารถใช้ฟังก์ชันอื่นๆ ได้เช่นกัน
- เราสามารถใช้ “**sum**” เพื่อหาผลรวมน้ำหนักทั้งหมด
- “**min**” เพื่อค้นหาน้ำหนักต่ำสุด
- “**max**” เพื่อค้นหาน้ำหนักสูงสุด
- หรือ “**count**” เพื่อค้นหาจำนวนสัตว์แต่ละประเภท

These two lines of code group the animals then apply the mean function to the weight column.

[ ]:
```
# Group by animal category
animal_groups = df.groupby("animal")
# Apply mean function to wieght column
animal_groups['weight'].mean()
```

[ ]:
```
df.groupby("animal").size()
```

[ ]:
```
df.groupby("animal").count()
```

[ ]:
```
df.groupby("animal")["weight"].count()
```

```
[ ]: df.groupby("weight")["animal"].count()
```

```
[ ]: df.groupby("weight")["animal"].max()
```

2.1 Next Example

```
[ ]: import pandas as pd

reviews = pd.read_csv("datasets/winemag-data-52.csv", index_col=0)
reviews
```

```
[ ]: #เช็คว่า DataFrame มีคอลัมน์อะไรบ้าง
reviews.columns
```

```
[ ]: #แสดงผลทางสถิติโดยยึด country เป็นหลัก
reviews.groupby('country').describe()
```

```
[ ]: reviews.groupby('country').size()
```

```
[ ]: #นับจำนวนทุกคอลัมน์โดยยึด country เป็นหลัก
reviews.groupby('country').count()
```

```
[ ]: reviews.groupby('points').size()
```

```
[ ]: #นับจำนวนทุกคอลัมน์โดยยึด points เป็นหลัก
reviews.groupby('points').count()
```

```
[ ]: #นับจำนวนคอลัมน์ price โดยยึด points เป็นหลัก
reviews.groupby('points').price.count()
```

```
[ ]: #นับจำนวนคอลัมน์ country โดยยึด points เป็นหลัก
reviews.groupby('points').country.count()
```

```
[ ]: #นับจำนวนคอลัมน์ country แยกแต่ละ value(ประเทศ) โดยยึด points เป็นหลัก
reviews.groupby('points').country.value_counts()
```

```
[ ]: #นับจำนวนคอลัมน์ price แยกแต่ละ value(ราคา) โดยยึด points เป็นหลัก
reviews.groupby('points').price.value_counts()
```

`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:

[ ]:
```
#แสดงผลค่าน้อยที่สุดในคอลัมน์ price โดยยึด points เป็นหลัก
reviews.groupby('points').price.min()
```

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:

2.1.1 การใช้ apply() กับ groupby()

เราสามารถใช้ apply() เพื่อทำงานกับข้อมูลในแต่ละกลุ่มได้อย่างยืดหยุ่น

[ ]:
```
#แสดงผลชื่อแรกในคอลัมน์ title โดยยึด points เป็นหลัก
reviews.groupby('points').apply(lambda df: df.title.iloc[0])
```

[ ]:
```
#แสดงผลชื่อแรกในคอลัมน์ title โดยยึด winery เป็นหลัก(เลือกชื่อไวน์รายการแรกที่รีวิวจากแต่ละโรงไวน์)
reviews.groupby('winery').apply(lambda df: df.title.iloc[0])
```

[ ]:
```
reviews.groupby('winery').title.count()
```

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:

[ ]:
```
#เลือกไวน์ที่คะแนนดีที่สุดตามประเทศ
reviews.groupby(['country']).apply(lambda df: df.loc[df.points.idxmax()])
```

[ ]:
```
#เลือกไวน์ที่คะแนนดีที่สุดตามจังหวัด
reviews.groupby(['province']).apply(lambda df: df.loc[df.points.idxmax()])
```

[ ]:
```
#สามารถจัดกลุ่มได้มากกว่าหนึ่งคอลัมน์
#ตัวอย่างเช่น เราจะเลือกไวน์ที่คะแนนดีที่สุดตามประเทศและจังหวัดได้อย่างไร:
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:

2.1.2 การใช้ agg() กับ groupby()

ฟังก์ชัน agg() (Aggregate functionคือฟังก์ชันสรุปผล เช่น len,min,max,sum,count,mean,median) ช่วยให้เราสามารถทำการวิเคราะห์หลายรูปแบบพร้อมกัน

[ ]:
```
#เราสามารถใช้คือ agg()
reviews.groupby(['country']).price.agg([len, min, max])
```

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

3 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:

เมื่อค่าของคอลัมน์เดียวไม่เพียงพอที่จะระบุแถวได้อย่างชัดเจน (เช่น ระเบียนหลายรายการในวันที่เดียวกัน หมายความว่าวันที่เพียงอย่างเดียวไม่เหมาะเป็นดัชนี) เมื่อข้อมูลมีลำดับชั้นเชิงตรรกะ ซึ่งหมายความว่ามีมิติหรือ "ระดับ" หลายระดับ นอกจากโครงสร้างแล้ว ดัชนีหลายดัชนียังช่วยให้เรียกค้นข้อมูลที่ซับซ้อนในหน่วยความจำได้ค่อนข้างง่าย

[ ]:
```
countries_reviewed = reviews.groupby(['country', 'province']).title.agg([len])
countries_reviewed
```

[ ]:
```
countries_reviewed.index
```

[ ]:
```
type(countries_reviewed.index)
```

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()
```

[ ]:
```
type(countries_reviewed.reset_index())
```

4 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')
```

[ ]:
```
countries_reviewed.sort_values('len')
```

`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').iloc[::-1]
```

[ ]:
```
countries_reviewed.sort_values(by='len', ascending=False)
```

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

[ ]:
```
countries_reviewed.sort_index()
```

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

[ ]:
```
countries_reviewed.sort_values(by=['country', 'len'])
```

5 Your turn

If you haven't started the exercise, you can start now.
