


Aleksey Bilogur

Summary functions and maps reference

Notebook

Summary functions and maps reference

This is the reference component to the "Summary functions and maps" section of the Advanced Pandas tutorial. For the workbook, click here.

This section overlaps with the comprehensive Essential Basic Functionality section of the official `pandas` documentation.

In [1]:
```
import pandas as pd
pd.set_option('max_rows', 5)
import numpy as np
import numpy as np
reviews = pd.read_csv("../input/winemag-data-130k-v2.csv", index_col=0)
reviews.head()
```

Out[1]:

	country	description	designation	points	price	province	region_1	region_2	taster_name	taster_twitter_handle	
0	Italy	Aromas include tropical fruit, broom, brimston...	Vulkà Bianco	87	NaN	Sicily & Sardinia	Etna	NaN	Kerin O'Keefe	@kerinokeefe	Nic 20 Bia (Et
1	Portugal	This is ripe and fruity, a wine that is smooth...	Avidagos	87	15.0	Douro	NaN	NaN	Roger Voss	@vossroger	Qu Avi 20 Avi Re (Do
2	US	Tart and snappy, the flavors of lime flesh and...	NaN	87	14.0	Oregon	Willamette Valley	Willamette Valley	Paul Gregutt	@paulgwine	Ra 20 Gr (W Val
3	US	Pineapple rind, lemon pith and orange blossom ...	Reserve Late Harvest	87	13.0	Michigan	Lake Michigan Shore	NaN	Alexander Peartree	NaN	St. 20 Re Lat Ha Rie
4	US	Much like the regular bottling from 2012, this...	Vintner's Reserve Wild Child Block	87	65.0	Oregon	Willamette Valley	Willamette Valley	Paul Gregutt	@paulgwine	Sw Ch 20 Vir Re Wi Ch

Summary functions

`pandas` provides many simple "summary functions" (not an official name) which restructure the data in some useful way. For example, consider the `describe` method:

In [2]:
```
reviews.points.describe()
```

Out[2]:
```
count    129971.000000
mean         88.447138
             ...
75%          91.000000
max         100.000000
Name: points, Length: 8, dtype: float64
```

This method generates a high-level summary of the attributes of the given column. It is type-aware, meaning that its output changes based on the `dtype` of the input. The output above only makes sense for numerical data; for string data here's what we get:

In [3]:
```
reviews.taster_name.describe()
```

Out[3]:
```
count         103727
unique            19
top       Roger Voss
freq           25514
Name: taster_name, dtype: object
```

If you want to get some particular simple summary statistic about a column in a `DataFrame` or a `Series`, there is usually a handful `pandas` function that makes it happen. For example, to see the mean of the points allotted (e.g. how well an averagely rated wine does), we can use the `mean` function:

In [4]:
```
reviews.points.mean()
```

Out[4]:
```
88.44713820775404
```

To see a list of unique values we can use the `unique` function:

In [5]:
```
reviews.taster_name.unique()
```

Out[5]:
```
array(['Kerin O’Keefe', 'Roger Voss', 'Paul Gregutt', 'Alexander Peartree',
       'Michael Schachner', 'Anna Lee C. Iijima', 'Virginie Boone',
       'Matt Kettmann', nan, 'Sean P. Sullivan', 'Jim Gordon',
       'Joe Czerwinski', 'Anne Krebiehl\xa0MW', 'Lauren Buzzeo',
       'Mike DeSimone', 'Jeff Jenssen', 'Susan Kostrzewa', 'Carrie Dykes',
       'Fiona Adams', 'Christina Pickard'], dtype=object)
```

To see a list of unique values *and* how often they occur in the dataset, we can use the `value_counts` method:

In [6]:
```
reviews.taster_name.value_counts()
```

Out[6]:
```
Roger Voss           25514
Michael Schachner    15134
                     ...
Fiona Adams             27
Christina Pickard        6
Name: taster_name, Length: 19, dtype: int64
```

Maps

A "map" is a term, borrowed from mathematics, for a function that takes one set of values and "maps" them to another set of values. In data science we often have a need for creating new representations from existing data, or for transforming data from the format it is in now to the format that we want it to be in later. Maps are what handle this work, making them extremely important for getting your work done!

There are two mapping functions that you will use often. The `Series` `map` is the first, and slightly simpler one. For example, suppose that we wanted to remean the scores the wines recieved to 0. We can do this as follows:

In [7]:
```
review_points_mean = reviews.points.mean()
reviews.points.map(lambda p: p - review_points_mean)
```

Out[7]:
```
0        -1.447138
1        -1.447138
            ...
129969    1.552862
129970    1.552862
Name: points, Length: 129971, dtype: float64
```

`map` takes every value in the column it is being called on and converts it some new value using a function you provide it.

`map` takes a `Series` as input. The `DataFrame` `apply` function can be used to do the same thing *across* columns, on the level of the entire dataset. Thus `apply` takes a `DataFrame` as input.

In [8]:
```
def remean_points(srs):
    srs.points = srs.points - review_points_mean
    return srs

reviews.apply(remean_points, axis='columns')
```

Out[8]:

	country	description	designation	points	price	province	region_1	region_2	taster_name	taster_twitter_handle
0	Italy	Aromas include tropical fruit, broom, brimston...	Vulkà Bianco	-1.447138	NaN	Sicily & Sardinia	Etna	NaN	Kerin O'Keefe	@kerinokeefe
1	Portugal	This is ripe and fruity, a wine that is smooth...	Avidagos	-1.447138	15.0	Douro	NaN	NaN	Roger Voss	@vossroger
...	...	...	...	...	...	...	...	...	...	...
129969	France	A dry style of Pinot Gris, this is crisp with ...	NaN	1.552862	32.0	Alsace	Alsace	NaN	Roger Voss	@vossroger
129970	France	Big, rich and off-dry, this is powered by inte...	Lieu-dit Harth Cuvée Caroline	1.552862	21.0	Alsace	Alsace	NaN	Roger Voss	@vossroger

129971 rows × 13 columns

`pandas` provides many common mapping operations as built-ins. For example, here's a faster way of remeaning our points column:

In [9]:
```
review_points_mean = reviews.points.mean()
reviews.points - review_points_mean
```

Out[9]:
```
0        -1.447138
1        -1.447138
            ...
129969    1.552862
129970    1.552862
Name: points, Length: 129971, dtype: float64
```

In this code we are performing an operation between a lot of values on the left-hand side (everything in the `Series`) and a single value on the right-hand side (the mean value). `pandas` looks at this expression and figures out that we must mean to subtract that mean value from every value in the dataset.

`pandas` will also understand what to do if we perform these operations between `Series` of equal length. For example, an easy way of combining country and region information in the dataset would be to do the following:

In [10]:
```
reviews.country + " - " + reviews.region_1
```

Out[10]:
```
0            Italy - Etna
1                     NaN
               ...
129969    France - Alsace
129970    France - Alsace
Length: 129971, dtype: object
```

These operators are faster than the `map` or `apply` because they uses speed ups built into `pandas`. All of the standard Python operators (`>`, `<`, `==`, and so on) work in this manner.

However, they are not as flexible as `map` or `apply`, which can do more advanced things, like applying conditional logic, which cannot be done with addition and subtraction alone.