Summary Functions and Maps

Extract insights from your data.

Tutorial Data

Learn Tutorial
Pandas

Course step
3 of 6

Introduction

In the last tutorial, we learned how to select relevant data out of a DataFrame or Series. Plucking the right data out of our data representation is critical to getting work done, as we demonstrated in the exercises.

However, the data does not always come out of memory in the format we want it in right out of the bat. Sometimes we have to do some more work ourselves to reformat it for the task at hand. This tutorial will cover different operations we can apply to our data to get the input "just right".

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

In [2]:
```
reviews
```

	country	description	designation	points	price	province	region_1	region_2	taster_name	taster_twitter_handle	title	vari
0	Italy	Aromas include tropical fruit, broom, brimston...	Vulkà Bianco	87	NaN	Sicily & Sardinia	Etna	NaN	Kerin O'Keefe	@kerinokeefe	Nicosia 2013 Vulkà Bianco (Etna)	Whi
1	Portugal	This is ripe and fruity, a wine that is smooth...	Avidagos	87	15.0	Douro	NaN	NaN	Roger Voss	@vossroger	Quinta dos Avidagos 2011 Avidagos Red (Douro)	Port
...	...	...	...	...	...	...	...	...	...	...	...	...
129969	France	A dry style of Pinot Gris, this is crisp with ...	NaN	90	32.0	Alsace	Alsace	NaN	Roger Voss	@vossroger	Domaine Marcel Deiss 2012 Pinot Gris (Alsace)	Pinc
129970	France	Big, rich and off-dry, this is powered by inte...	Lieu-dit Harth Cuvée Caroline	90	21.0	Alsace	Alsace	NaN	Roger Voss	@vossroger	Domaine Schoffit 2012 Lieu-dit Harth Cuvée Car...	Gew

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 [3]:
```
reviews.points.describe()
```

Out[3]:
```
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 data type of the input. The output above only makes sense for numerical data; for string data here's what we get:

In [4]:
```
reviews.taster_name.describe()
```

Out[4]:
```
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 helpful 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 [5]:
```
reviews.points.mean()
```

Out[5]:
```
88.44713820775404
```

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

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

Out[6]:
```
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 [7]:
```
reviews.taster_name.value_counts()
```

Out[7]:
```
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 methods that you will use often.

`map()` is the first, and slightly simpler one. For example, suppose that we wanted to remean the scores the wines received to 0. We can do this as follows:

In [8]:
```
review_points_mean = reviews.points.mean()
reviews.points.map(lambda p: p - review_points_mean)
```

Out[8]:
```
0        -1.447138
1        -1.447138
              ...
129969    1.552862
129970    1.552862
Name: points, Length: 129971, dtype: float64
```

The function you pass to `map()` should expect a single value from the Series (a point value, in the above example), and return a transformed version of that value. `map()` returns a new Series where all the values have been transformed by your function.

`apply()` is the equivalent method if we want to transform a whole DataFrame by calling a custom method on each row.

In [9]:
```
def remean_points(row):
    row.points = row.points - review_points_mean
    return row

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

Out[9]:
Out[9]:

	country	description	designation	points	price	province	region_1	region_2	taster_name	taster_twitter_handle	title
0	Italy	Aromas include tropical fruit, broom, brimston...	Vulkà Bianco	-1.447138	NaN	Sicily & Sardinia	Etna	NaN	Kerin O'Keefe	@kerinokeefe	Nicosia 2013 Vulkà Bianco (Etna)
1	Portugal	This is ripe and fruity, a wine that is smooth...	Avidagos	-1.447138	15.0	Douro	NaN	NaN	Roger Voss	@vossroger	Quinta dos Avidagos 2011 Avidagos Red (Douro)
...	...	...	...	...	...	...	...	...	...	...	...
129969	France	A dry style of Pinot Gris, this is crisp with ...	NaN	1.552862	32.0	Alsace	Alsace	NaN	Roger Voss	@vossroger	Domaine Marcel Deiss 2012 Pinot Gris (Alsace)
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	Domaine Schoffit 2012 Lieu-dit Harth Cuvée Car...

If we had called `reviews.apply()` with `axis='index'`, then instead of passing a function to transform each row, we would need to give a function to transform each *column*.

Note that `map()` and `apply()` return new, transformed Series and DataFrames, respectively. They don't modify the original data they're called on. If we look at the first row of `reviews`, we can see that it still has its original `points` value.

In [10]:
```
reviews.head(1)
```

Out[10]:

	country	description	designation	points	price	province	region_1	region_2	taster_name	taster_twitter_handle	title	variety	win
0	Italy	Aromas include tropical fruit, broom, brimston...	Vulkà Bianco	87	NaN	Sicily & Sardinia	Etna	NaN	Kerin O'Keefe	@kerinokeefe	Nicosia 2013 Vulkà Bianco (Etna)	White Blend	Nic

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

In [11]:
```
review_points_mean = reviews.points.mean()
reviews.points - review_points_mean
```

Out[11]:
```
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 [12]:
```
reviews.country + " - " + reviews.region_1
```

Out[12]:
```
0            Italy - Etna
1                     NaN
                 ...
129969    France - Alsace
129970    France - Alsace
Length: 129971, dtype: object
```

These operators are faster than `map()` or `apply()` because they use 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.