Pandas: 4th lesson – Summary Functions & Maps

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:

reviews.points.describe()

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:

reviews.taster\_name.describe()

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:

reviews.points.mean()

88.44713820775404

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

reviews.taster\_name.unique()

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:

reviews.taster\_name.value\_counts()

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:

review\_points\_mean = reviews.points.mean()

reviews.points.map(lambda p: p - review\_points\_mean)

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:

def remean\_points(row):

row.points = row.points - review\_points\_mean

return row

reviews.apply(remean\_points, axis='columns')

* 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:

reviews.head(1)

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

review\_points\_mean = reviews.points.mean()

reviews.points - review\_points\_mean

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:

reviews.country + " - " + reviews.region\_1

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.