Data "wrangling"

Sunday, May 5, 2019 10:44 AM

You might ask why it's necessary for a data scientist to be able to program at all. Why isn't every necessary program already written?

The answer is that in general, a data scientist doesn't do much programming; instead, much of the work is done by programs written by others.

However, there is one important exception: what I might call "***data wrangling***".

- The data is often provided in an arbitrary format.
- Analysis tools require a very specific format as a precondition.
- You must write code that transforms one kind of data into another.
- This can be as simple as reformatting an array, or as complex as mining data from text!

An oversimplified example

- You have a budget as text.
- You want the total for the budget.

Say the budget is described as follows:

```
Airfare 300.00
Hotel 200.00
Food 100.00
```

…

There are several kinds of text input. This is sometimes called "text with variable-width columns and delimiters".

In python, we might write this as

```
budget = """Airfare 300.00
Hotel 200.00
Food 100.00"""
```

(""" … """ is a multi-line string)

What we want is a list of the items, e.g.,

```
costs = [300.00, 200.00, 100.00]
```

Then we could sum these up via:

```
sum = 0
for c in costs:
    sum = sum + c
print(sum)
```

But to get to this situation requires some data transformation.

From one string to array of list:

```
lines = budget.split('\n')
```

This splits the one multi-line string into one string per line, in a list. The line delimiter is '\n'.

From list of lines to list of costs:

```
costs = []
for l in lines:
```

```
        thing, cost = l.split(" ")
        number = float(cost)
        costs.append(number)
```

This splits each line into two things, separated by space, and then treats the second one as a number.

Syntax: `v1, v2, … = string.split('<delimiter>')`

And now sum it up:

```
sum = 0
for c in costs:
   sum += c
```

And the answer, as we would expect, is 600.00.

Notice that I did this by proceeding in steps and doing one step at a time. To get to an end-result.

1. Plan out the end-result. What should it look like?
2. Figure out intermediate transformations that will get you to that result.
3. Do one at a time.

In this case, I knew that splitting on '\n' would get me lines, and that splitting on ' ' would get me words. So I used that to get the result.

Here is a notebook that computes this:

```
In [1]: budget = """Airfare 300.00
        Hotel 200.00
        Food 100.00"""
        budget
Out[1]: 'Airfare 300.00\nHotel 200.00\nFood 100.00'

In [2]: lines = budget.split('\n')
        lines
Out[2]: ['Airfare 300.00', 'Hotel 200.00', 'Food 100.00']

In [3]: costs = []
        for l in lines:
            thing, cost = l.split(" ")
            number = float(cost)
            costs.append(number)
        costs
Out[3]: [300.0, 200.0, 100.0]

In [4]: sum = 0
        for c in costs:
            sum += c
        sum
Out[4]: 600.0
```

But wait, what if our input is not that behaved?

- So far, we've considered input that was highly structured.
- What would we do if it weren't highly structured?
- We might have to resort to text mining instead.