Pandas: 2nd lesson – Indexing, Selecting & Assigning

Native accessors:

* Native Python objects provide good ways of indexing data. Pandas carries all of these over, which helps make it easy to start with.
* In Python, we can access the property of an object by accessing it as an attribute. A book object, for example, might have a title property, which we can access by calling book.title. Columns in a pandas DataFrame work in much the same way.
* If we have a Python dictionary, we can access its values using the indexing ([]) operator.
* These are the two ways of selecting a specific Series out of a DataFrame. Neither of them is more or less syntactically valid than the other, but the indexing operator [] does have the advantage that it can handle column names with reserved characters in them (e.g. if we had a country providence column, reviews.country providence wouldn't work).

Indexing in pandas:

The indexing operator and attribute selection are nice because they work just like they do in the rest of the Python ecosystem. As a novice, this makes them easy to pick up and use. However, pandas has its own accessor operators, loc and iloc. For more advanced operations, these are the ones you're supposed to be using.

Manipulating the index:

Label-based selection derives its power from the labels in the index. Critically, the index we use is not immutable. We can manipulate the index in any way we see fit. The set\_index() method can be used to do the job.

Conditional selection:

So far, we've been indexing various strides of data, using structural properties of the DataFrame itself. To do interesting things with the data, however, we often need to ask questions based on conditions.