Exploring Data with pandas: Takeaways

Syntax

USING ILOC[] TO SELECT BY INTEGER POSITION

- Selecting a value:

```
third_row_first_col = df.iloc[2,0]
```

- Selecting a row:

```
second_row = df.iloc[1]
```

CREATING BOOLEAN MASKS USING PANDAS METHODS

- String method that returns Boolean series object:

```
is_california = usa["hq_location"].str.endswith("CA")
```

- Filtering using Boolean series object:

```
df_where_filter_true = usa[is_california]
```

- Selecting only the non-null values in a column:

```
f500[f500["previous_rank"].notnull()]
```

BOOLEAN OPERATORS

- Multiple required filtering criteria:

```
filter_big_rev_neg_profit = (f500["revenues"] > 100000) & (f500["profits"] < 0)
```

- Multiple optional filtering criteria:

```
filter_big_rev_neg_profit = (f500["revenues"] > 100000) | (f500["profits"] < 0)
```

Concepts

- To select values by axis labels, use `loc[]`. To select values by integer locations, use `iloc[]`. When the label for an axis is just its integer position, these methods can be mostly used interchangeably.
- To take advantage of vectorization in pandas but think and speak in filtering criteria (instead of integer index values), you'll find yourself expressing many computations as Boolean masks and filtering series and dataframes. Because using a loop doesn't take advantage of vectorization, it's important to avoid doing so unless you absolutely have to. Boolean operators are a powerful technique to further take advantage of vectorization when filtering because you're able to express more granular filters.

Resources

- Boolean Indexing
- iloc vs loc