Getting Started with pandas

Part 6

Essential Functionality

Part 4

Sorting and Ranking

- Sorting a dataset by some criterion is another important built-in operation.
- To sort lexicographically by row or column index, use the `sort_index` method, which returns a new, sorted object:

```
In [167]: obj = pd.Series(range(4), index=['d', 'a', 'b', 'c'])

In [168]: obj

Out[168]: d    0
          a    1
          b    2
          c    3
          dtype: int64

In [169]: obj.sort_index()

Out[169]: a    1
          b    2
          c    3
          d    0
          dtype: int64
```

- With a DataFrame, you can sort by index on either axis:

```
In [170]: frame = pd.DataFrame(np.arange(8).reshape((2, 4)),
                               index=['three', 'one'],
                               columns=['d', 'a', 'b', 'c'])

In [171]: frame
```

Out[171]:

	d	a	b	c
three	0	1	2	3
one	4	5	6	7

```
In [172]: frame.sort_index()
```

Out[172]:

	d	a	b	c
one	4	5	6	7
three	0	1	2	3

```
In [173]: frame.sort_index(axis=1)
```

Out[173]:

	a	b	c	d
three	1	2	3	0
one	5	6	7	4

- The data is sorted in ascending order by default, but can be sorted in descending order, too:

```
In [174]: frame.sort_index(axis=1, ascending=False)
```

Out[174]:

	d	c	b	a
three	0	3	2	1
one	4	7	6	5

- To sort a Series by its values, use its `sort_values` method:

```
In [175]: obj = pd.Series([4, 7, -3, 2])

In [176]: obj

Out[176]: 0    4
          1    7
          2   -3
          3    2
          dtype: int64

In [177]: obj.sort_values()

Out[177]: 2   -3
          3    2
          0    4
          1    7
          dtype: int64
```

- Any missing values are sorted to the end of the Series by default:

```
In [178]: obj = pd.Series([4, np.nan, 7, np.nan, -3, 2])

In [179]: obj

Out[179]: 0    4.0
          1    NaN
          2    7.0
          3    NaN
          4   -3.0
          5    2.0
          dtype: float64

In [180]: obj.sort_values()

Out[180]: 4   -3.0
          5    2.0
          0    4.0
          2    7.0
          1    NaN
          3    NaN
          dtype: float64
```

- When sorting a DataFrame, you can use the data in one or more columns as the sort keys.
- To do so, pass one or more column names to the `by` option of `sort_values`:

```
In [181]: frame = pd.DataFrame({'b': [4, 7, -3, 2], 'a': [0, 1, 0, 1]})

In [182]: frame
Out[182]:
```

	b	a
0	4	0
1	7	1
2	-3	0
3	2	1

```
In [183]: frame.sort_values(by='b')
Out[183]:
```

	b	a
2	-3	0
3	2	1
0	4	0
1	7	1

- To sort by multiple columns, pass a list of names:

```
In [184]: frame.sort_values(by=['a', 'b'])
```

Out[184]:

	b	a
2	-3	0
0	4	0
3	2	1
1	7	1

- *Ranking* assigns ranks from one through the number of valid data points in an array.
- The `rank` methods for Series and DataFrame are the place to look; by default rank breaks ties by assigning each group the mean rank:

```
In [185]: obj = pd.Series([7, -5, 7, 4, 2, 0, 4])

In [186]: obj
Out[186]: 0    7
          1   -5
          2    7
          3    4
          4    2
          5    0
          6    4
          dtype: int64

In [187]: obj.rank()
Out[187]: 0    6.5
          1    1.0
          2    6.5
          3    4.5
          4    3.0
          5    2.0
          6    4.5
          dtype: float64
```

- Ranks can also be assigned according to the order in which they're observed in the data:

```
In [188]: obj
Out[188]: 0    7
          1   -5
          2    7
          3    4
          4    2
          5    0
          6    4
          dtype: int64

In [189]: obj.rank(method='first')
Out[189]: 0    6.0
          1    1.0
          2    7.0
          3    4.0
          4    3.0
          5    2.0
          6    5.0
          dtype: float64
```

- You can rank in descending order, too:

```
In [190]: obj
Out[190]: 0    7
          1   -5
          2    7
          3    4
          4    2
          5    0
          6    4
          dtype: int64

In [191]: # Assign tie values the maximum rank in the group
          obj.rank(ascending=False, method='max')
Out[191]: 0    2.0
          1    7.0
          2    2.0
          3    4.0
          4    5.0
          5    6.0
          6    4.0
          dtype: float64
```

- Tie-breaking methods with rank

Method	Description
`'average'`	Default: assign the average rank to each entry in the equal group
`'min'`	Use the minimum rank for the whole group
`'max'`	Use the maximum rank for the whole group
`'first'`	Assign ranks in the order the values appear in the data
`'dense'`	Like `method='min'`, but ranks always increase by 1 in between groups rather than the number of equal elements in a group

- DataFrame can compute ranks over the rows or the columns:

```
In [192]: frame = pd.DataFrame({'b': [4.3, 7, -3, 2], 'a': [0, 1, 0, 1],
                                'c': [-2, 5, 8, -2.5]})
```

```
In [193]: frame
```

Out[193]:

	b	a	c
0	4.3	0	-2.0
1	7.0	1	5.0
2	-3.0	0	8.0
3	2.0	1	-2.5

```
In [194]: frame.rank(axis='columns')
```

Out[194]:

	b	a	c
0	3.0	2.0	1.0
1	3.0	1.0	2.0
2	1.0	2.0	3.0
3	3.0	2.0	1.0

Axis Indexes with Duplicate Labels

- Up until now all of the examples we've looked at have had unique axis labels (index values).
- While many pandas functions (like `reindex`) require that the labels be unique, it's not mandatory.
- Let's consider a small Series with duplicate indices:

```
In [195]: obj = pd.Series(range(5), index=['a', 'a', 'b', 'b', 'c'])
          obj

Out[195]: a    0
          a    1
          b    2
          b    3
          c    4
          dtype: int64
```

- The index's `is_unique` property can tell you whether its labels are unique or not:

```
In [196]: obj.index.is_unique
Out[196]: False
```

- Data selection is one of the main things that behaves differently with duplicates.
- Indexing a label with multiple entries returns a Series, while single entries return a scalar value:

```
In [197]: obj['a']

Out[197]: a    0
          a    1
          dtype: int64

In [198]: obj['c']

Out[198]: 4
```

- This can make your code more complicated, as the output type from indexing can vary based on whether a label is repeated or not.

- The same logic extends to indexing rows in a DataFrame:

```
In [199]: df = pd.DataFrame(np.random.randn(4, 3), index=['a', 'a', 'b', 'b'])
          df
```

Out[199]:

	0	1	2
a	0.274992	0.228913	1.352917
a	0.886429	-2.001637	-0.371843
b	1.669025	-0.438570	-0.539741
b	0.476985	3.248944	-1.021228

```
In [200]: df.loc['b']
```

Out[200]:

	0	1	2
b	1.669025	-0.438570	-0.539741
b	0.476985	3.248944	-1.021228