Sort and Rank in Pandas

In this lesson, the rank and sort functions of pandas objects are explained.

We'll cover the following

Sort

Sort, as the name suggests, simply sorts a `Series` object in ascending order. The `pandas` package provides functions to sort both the indexes and their values. The following functions are used for this:

- `sort_index()`: This function sorts the *indexes* in ascending order. It works on the alphabetic, numeric, and alphanumeric *indexes*. The non-numeric index values are treated as corresponding ASCII codes and sorted accordingly.

```
import pandas as pd

# Numeric indexes
print("Numeric Indexes Sorted")
srs1 = pd.Series([1,2,3], index = ['3', '1', '2'])
print(srs1.sort_index())

# Alphabetic indexes
print("\nAlphabetic Indexes Sorted")
srs2 = pd.Series([1,2,3], index = ['B', 'C', 'A'])
print(srs2.sort_index())

# Alhanumeric indexes
print("\nAlphanumeric Indexes Sorted")
srs3 = pd.Series([1,2,3], index = ['3', '2', 'A'])
print(srs3.sort_index())
```

All 3 types of *indexes* are sorted in the above example. It can be seen from the output that, by sorting the indexes, the values of the `Series` get all jumbled up. So, there is another function for sorting the values, too.

- `sort_values()` This function sorts the values of a `Series`. This can sort both the alphabetic and numeric *values*. Just like the `sort_index()` method, the non-numeric values are treated as *ASCII* characters.

```
import pandas as pd
import numpy as np

# Numeric values
print("Random Numeric Values Sorted")
srs1 = pd.Series(np.random.randn(5))
print(srs1.sort_values())

# Alphabetic values
print("\nAlphabetic Values Sorted")
srs2 = pd.Series(['D', 'A', 'E', 'C', 'B'])
print(srs2.sort_values())
```

In the output, it can be observed that just like the `sort_index()` method mixed up the *values*, the `sort_values()` method has mixed up the *indexes*.

> **Note**: At one point, either *indexes* or *values*, only one can be sorted. It is decided by the *user* based on the type of problem.

Rank

Rank is basically the positioning of *indexes* according to the sorted values of a series. In simpler words, rank is assigned to the *indexes* from **1 to n**, based on the value corresponding to the *index*. This tells us which place the value takes in the series if the list was sorted. The following illustration might make it more clear:

Rank of indexes when values are not sorted

Rank and position of indexes after sorting the values


```
import pandas as pd
import numpy as np
```

```
srs = pd.Series(np.random.randn(5))

print("The series:")
print(srs)
# Ranks before sorting
print("\nRanks before sorting:")
print(srs.rank())

srs = srs.sort_values()
# Ranks after sorting
print("\nRanks after sorting:")
print(srs.rank())
```

In the output, the first column represents the *index*, and the second column represents the corresponding rank of that *index*. If observed closely, it can be seen that ranks are the same, before and after sorting. However, after sorting, the indexes are placed according to their ranks in ascending order.

In the next lesson, situations with missing data are discussed.