Additional Functions and Features

In this lesson, some important functions of pandas are explored.

We'll cover the following

- Pandas functions
- Multilevel indexing
  - Multilevel indexing in Series
  - Multilevel indexing in DataFrame

Pandas functions

The following functions are explained:

1. `sum(axis=0)` : This function calculates the sum of each column of a `DataFrame`.
2. `sum(axis=1)` : This function calculates the sum of each row of a `DataFrame`.

```
import numpy as np
import pandas as pd
# Declaring DataFrame
df = pd.DataFrame(np.arange(9).reshape(3,3), index=['A', 'B', 'C'], columns=['A', 'B', 'C'])

print("The DataFrame")
print(df)

print("\nThe sum of each Column:")
print(df.sum(axis=0))

print("\nThe sum of each Row:")
print(df.sum(axis=1))
```

The sum of the column and row elements are clearly visible in the output.

3. `min(axis=0)` : This function returns the minimum value from each column.
4. `min(axis=1)` : This function returns the minimum value from each row.

```
import numpy as np
import pandas as pd
# Declaring DataFrame

df = pd.DataFrame(np.arange(9).reshape(3,3), index=['A', 'B', 'C'], columns=['A', 'B', 'C'])

print("The DataFrame")
print(df)

print("\nThe minimum from each Column:")
print(df.min(axis=0))

print("\nThe minimum from each Row:")
print(df.min(axis=1))
```

It can be clearly seen from the output that the minimum value from both row and column is printed using the function.

5. `idxmin(axis=0)` : This function returns the *index* with minimum value from every column.

6. `idxmin(axis=1)` : This function returns the column with minimum value from every *index*.

```
import numpy as np
import pandas as pd
# Declaring DataFrame
df = pd.DataFrame(np.arange(9).reshape(3,3), index=['A', 'B', 'C'], columns=['Col1', 'Col2', 'Col3

print("The DataFrame")
print(df)

print("\nThe minimum value in each Column is at index:")
print(df.idxmin(axis=0))

print("\nThe minimum value at each index is at Column:")
print(df.idxmin(axis=1))
```

Multilevel indexing

Until now, only one level of indexing was mentioned for both the `Series` and `DataFrame`, but the *indexes* can also be multileveled. This is when one *index* refers to one or more *indexes*, and those *indexes* further refer to values. This can be useful when dealing with different kinds of data.

Multilevel indexing in `Series`

Multilevel indexing in `Series`

The following example shows how to make multiple *indexes* in a `Series`.

```
import numpy as np
import pandas as pd
# Declaring a multilevel indexed Series
srs = pd.Series(np.arange(5), index=[['A','A','B','B','B'],[1,2,3,4,5]])

print("The multileveled index in Series:")
print(srs)

print("\nThe A index:")
print(srs['A']) # Fetching elements at index named A

print("\nThe B index:")
print(srs['B']) # Fetching elements at index named B
```

As seen on **line 4**, the syntax is the same as the single level *index* except that now *two* lists are passed in the *index* parameter.

The first list parameter is the outer *index*, and the second is the inner *index*. The remaining rules of defining a `Series` remains the same.

Multilevel indexing in `DataFrame`

A `DataFrame` has multiple levels of *columns* as well as multiple levels of *indexes*. The following example explains this behavior.

```
import numpy as np
import pandas as pd
# Declaring a multilevel indexed DataFrame
df = pd.DataFrame(np.arange(25).reshape(5,5), index=[['A','A','A','B','B'], [1,2,3,4,5]],
                        columns=[['USA', 'Pak', 'Pak', 'UK','Ind'], ['Day', 'Day','Night', 'Night'

print("The multileveled index in DataFrame:\n")
print(df)
```

Just like `Series`, two lists are also passed to the *column* parameter as well as the *index* parameter. The same rules for multilevel indexing in `Series` also apply to `DataFrame`.

That's it for the `Pandas` package of Python. Next, a challenge to test your `pandas` skills awaits.