Pandas

Pandas is a very popular library for working with data (its goal is to be the most powerful and flexible open-source tool, and in our opinion, it has reached that goal). Data Frames are at the center of pandas. A Data Frame is structured like a table or spreadsheet. The rows and the columns both have indexes, and you can perform operations on rows or columns separately.

A pandas Data Frame can be easily changed and manipulated. Pandas has helpful functions for handling missing data, performing operations on columns and rows, and transforming data. If that wasn't enough, a lot of SQL functions have counterparts in pandas, such as join, merge, filter by, and group by. With all of these powerful tools, it should come as no surprise that pandas is very popular among data scientists.

Pandas Series and Data frames:

The *Series* is the core object of the pandas library. A pandas Series is very similar to a one-dimensional NumPy array, but it has additional functionality that allows values in the Series to be indexed using labels. A NumPy array does not have the flexibility to do this. This labeling is useful when you are storing pieces of data that have other data associated with them. Say you want to store the ages of students in an online course to eventually figure out the average student age. If stored in a NumPy array, you could only access these ages with the internal ndarray indices $0,1,2$.... With a Series object, the indices of values are set to $0,1,2$... by default, but you can customize the indices to be other values such as student names so an age can be accessed using a name. Customized indices of a Series are established by sending values into the Series constructor, as you will see below.

A Series holds items of any one data type and can be created by sending in a scalar value, Python list, dictionary, or ndarray as a parameter to the pandas Series constructor. If a dictionary is sent in, the keys may be used as the indices.

```
# Create a Series using a NumPy array of ages with the default numerical indices
ages = np.array([13,25,19])
series1 = pd.Series(ages)
print(series1)
```

```
0 | 13
1 | 25
2 | 19
dtype: int64
```

```
series1 = pd.Series(ages,index=['Emma', 'Swetha', 'Serajh']) # to change index
```

```
Emma   | 13
Swetha | 25
Serajh | 19
dtype: int64
```

Another important type of object in the pandas library is the *DataFrame*. This object is similar in form to a matrix as it consists of rows and columns. Both rows and columns can be indexed with integers or String names. One Data Frame can contain many different types of data types, but within a column, everything has to be the same data type. A column of a Data Frame is essentially a Series. All columns must have the same number of elements (rows).

There are different ways to fill a Data Frame such as with a CSV file, a SQL query, a Python list, or a dictionary. Here we have created a Data Frame using a Python list of lists. Each nested list represents the data in one row of the Data Frame. We use the keyword columns to pass in the list of our custom column names.

```
dataf = pd.DataFrame([
   ['John Smith' '123 Main St' 34]
   ['Jane Doe' '456 Maple Ave' 28]
   ['Joe Schmo' '789 Broadway' 51]
   ],
   Columns = ['name' 'address' 'age'])
```

This is how the Data Frame is displayed:

```
       name     |  address     |  age
0   | John Smith   | 123 Main St   |   34
1   | Jane Doe     | 456 Maple Ave |   28
2   | Joe Schmo    | 789 Broadway  |   51
```

The default row indices are 0,1,2..., but these can be changed. For example, they can be set to be the elements in one of the columns of the Data Frame. To use the names column as indices instead of the default numerical values, we can run the following command on our Data Frame:

```
dataf.set_index('name')
```

```
  name     |  address     |  age
John Smith   | 123 Main St   |   34
Jane Doe     | 456 Maple Ave |   28
Joe Schmo    | 789 Broadway  |   51
```

Data Frames are useful because they make it much easier to select, manipulate, and summarize data. Their tabular format (a table with rows and columns) also makes it easier to label, simpler to read, and easier to export data to and from a spreadsheet. Understanding the power of these new data structures is the key to unlocking many new avenues for data manipulation, exploration, and analysis!