CHAPTER 11

Pandas

Pandas is a specialized Python library for data analysis, especially on humongous datasets. It boasts easy-to-use functionality for reading and writing data, dealing with missing data, reshaping the dataset, and massaging the data by slicing, indexing, inserting, and deleting data variables and records. Pandas also has an important **groupBy** functionality for aggregating data for defined conditions – useful for plotting and computing data summaries for exploration.

Another key strength of Pandas is in re-ordering and cleaning time series data for time series analysis. In short, Pandas is the go-to tool for data cleaning and data exploration.

To use Pandas, first import the Pandas module:

```
import pandas as pd
```

Pandas Data Structures

Just like NumPy, Pandas can store and manipulate a multi-dimensional array of data. To handle this, Pandas has the **Series** and **DataFrame** data structures.

Series

The **Series** data structure is for storing a 1-D array (or vector) of data elements. A series data structure also provides labels to the data items in the form of an **index**. The user can specify this label via the **index** parameter in the **Series** function, but if the **index** parameter is left unspecified, a default label of 0 to one minus the size of the data elements is assigned.

E. Bisong, *Building Machine Learning and Deep Learning Models on Google Cloud Platform*,
https://doi.org/10.1007/978-1-4842-4470-8_11

Let us consider an example of creating a **Series** data structure.

```
# create a Series object
my_series = pd.Series([2,4,6,8], index=['e1','e2','e3','e4'])
# print out data in Series data structure
my_series
'Output':
e1    2
e2    4
e3    6
e4    8
dtype: int64
# check the data type of the variable
type(my_series)
'Output': pandas.core.series.Series
# return the elements of the Series data structure
my_series.values
'Output': array([2, 4, 6, 8])
# retrieve elements from Series data structure based on their assigned
indices
my_series['e1']
'Output': 2
# return all indices of the Series data structure
my_series.index
'Output': Index(['e1', 'e2', 'e3', 'e4'], dtype='object')
```

Elements in a Series data structure can be assigned the same indices.

```
# create a Series object with elements sharing indices
my_series = pd.Series([2,4,6,8], index=['e1','e2','e1','e2'])
# note the same index assigned to various elements
my_series
'Output':
e1    2
e2    4
e1    6
e2    8
```

```
dtype: int64
# get elements using their index
my_series['e1']
'Output':
e1    2
e1    6
dtype: int64
```

DataFrames

A DataFrame is a Pandas data structure for storing and manipulating 2-D arrays. A 2-D array is a table-like structure that is similar to an Excel spreadsheet or a relational database table. A DataFrame is a very natural form for storing structured datasets.

A DataFrame consists of rows and columns for storing records of information (in rows) across heterogeneous variables (in columns).

Let's see examples of working with DataFrames.

```
# create a data frame
my_DF = pd.DataFrame({'age': [15,17,21,29,25], \
            'state_of_origin':['Lagos', 'Cross River', 'Kano', 'Abia',
            'Benue']})
my_DF
'Output':
   age state_of_origin
0   15           Lagos
1   17     Cross River
2   21            Kano
3   29            Abia
4   25           Benue
```

We will observe from the preceding example that a DataFrame is constructed from a dictionary of records where each value is a **Series** data structure. Also note that each row has an **index** that can be assigned when creating the DataFrame, else the default from 0 to one off the number of records in the DataFrame is used. Creating an index manually is usually not feasible except when working with small dummy datasets.