```
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.

NumPy is frequently used together with Pandas. Let's import the NumPy library and use some of its functions to demonstrate other ways of creating a quick DataFrame.

```
import numpy as np

# create a 3x3 dataframe of numbers from the normal distribution
my_DF = pd.DataFrame(np.random.randn(3,3),\
            columns=['First','Second','Third'])
my_DF
'Output':
      First    Second     Third
0 -0.211218 -0.499870 -0.609792
1 -0.295363  0.388722  0.316661
2  1.397300 -0.894861  1.127306
# check the dimensions
my_DF.shape
'Output': (3, 3)
```

Let's examine some other operations with DataFrames.

```
# create a python dictionary
my_dict = {'State':['Adamawa', 'Akwa-Ibom', 'Yobe', 'Rivers', 'Taraba'], \
            'Capital':['Yola','Uyo','Damaturu','Port-Harcourt','Jalingo'], \
            'Population':[3178950, 5450758, 2321339, 5198716, 2294800]}
my_dict
'Output':
{'Capital': ['Yola', 'Uyo', 'Damaturu', 'Port-Harcourt', 'Jalingo'],
 'Population': [3178950, 5450758, 2321339, 5198716, 2294800],
 'State': ['Adamawa', 'Akwa-Ibom', 'Yobe', 'Rivers', 'Taraba']}
# confirm dictionary type
type(my_dict)
'Output': dict
# create DataFrame from dictionary
my_DF = pd.DataFrame(my_dict)
my_DF
```

```
'Output':
         Capital  Population      State
0           Yola     3178950    Adamawa
1            Uyo     5450758  Akwa-Ibom
2       Damaturu     2321339       Yobe
3  Port-Harcourt     5198716     Rivers
4        Jalingo     2294800     Taraba
# check DataFrame type
type(my_DF)
'Output': pandas.core.frame.DataFrame
# retrieve column names of the DataFrame
my_DF.columns
'Output': Index(['Capital', 'Population', 'State'], dtype='object')
# the data type of `DF.columns` method is an Index
type(my_DF.columns)
'Output': pandas.core.indexes.base.Index
# retrieve the DataFrame values as a NumPy ndarray
my_DF.values
'Output':
array([['Yola', 3178950, 'Adamawa'],
       ['Uyo', 5450758, 'Akwa-Ibom'],
       ['Damaturu', 2321339, 'Yobe'],
       ['Port-Harcourt', 5198716, 'Rivers'],
       ['Jalingo', 2294800, 'Taraba']], dtype=object)
# the data type of  `DF.values` method is an numpy ndarray
type(my_DF.values)
'Output': numpy.ndarray
```

In summary, a DataFrame is a tabular structure for storing a structured dataset where each column contains a **Series** data structure of records. Here's an illustration (Figure 11-1).

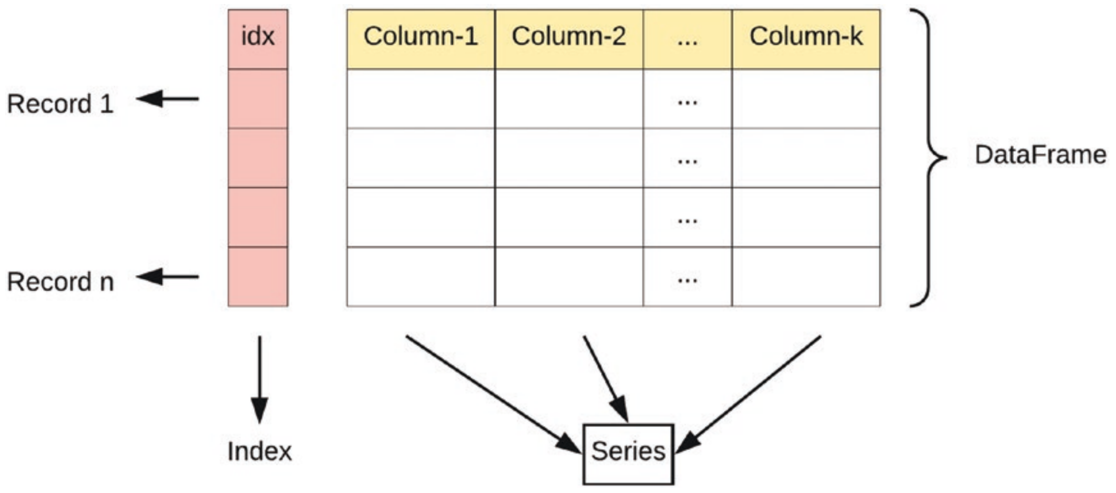

Figure 11-1. *Pandas data structure*

Let's check the data type of each column in the DataFrame.

```
my_DF.dtypes
'Output':
Capital       object
Population     int64
State         object
dtype: object
```

An **object** data type in Pandas represents **Strings**.

Data Indexing (Selection/Subsets)

Similar to NumPy, Pandas objects can index or subset the dataset to retrieve a specific sub-record of the larger dataset. Note that data indexing returns a new **DataFrame** or **Series** if a 2-D or 1-D array is retrieved. They do not, however, alter the original dataset. Let's go through some examples of indexing a Pandas DataFrame.

First let's create a dataframe. Observe the default integer indices assigned.

```
# create the dataframe
my_DF = pd.DataFrame({'age': [15,17,21,29,25], \
            'state_of_origin':['Lagos', 'Cross River', 'Kano', 'Abia',
            'Benue']})
```