```
'Output':
   age state_of_origin
0   15           Lagos
1   17     Cross River
2   21            Kano
3   29            Abia
4   25           Benue
# add a row to data frame
my_DF = my_DF.append(pd.Series([30 , 'Osun'], index=my_DF.columns), \
                                                    ignore_index=True)
my_DF
'Output':
   age state_of_origin
0   15           Lagos
1   17     Cross River
2   21            Kano
3   29            Abia
4   25           Benue
5   30            Osun
```

We observe that adding a new row involves passing to the **append** method, a **Series** object with the **index** attribute set to the columns of the main DataFrame. Since typically, in given datasets, the index is nothing more than the assigned defaults, we set the attribute **ignore_index** to create a new set of default index values with the new row(s).

Data Alignment

Pandas utilizes data alignment to align indices when performing some binary arithmetic operation on DataFrames. If two or more DataFrames in an arithmetic operation do not share a common index, a **NaN** is introduced denoting missing data. Let's see examples of this.

```
# create a 3x3 dataframe - remember randint(low, high, size)
df_A = pd.DataFrame(np.random.randint(1,10,[3,3]),\
            columns=['First','Second','Third'])
df_A
```

```
'Output':
   First  Second  Third
0      2       3      9
1      8       7      7
2      8       6      4
# create a 4x3 dataframe
df_B = pd.DataFrame(np.random.randint(1,10,[4,3]),\
            columns=['First','Second','Third'])
df_B
'Output':
   First  Second  Third
0      3       6      3
1      2       2      1
2      9       3      8
3      2       9      2
# add df_A and df_B together
df_A + df_B
'Output':
   First  Second  Third
0    5.0     9.0   12.0
1   10.0     9.0    8.0
2   17.0     9.0   12.0
3    NaN     NaN    NaN
# divide both dataframes
df_A / df_B
'Output':
      First  Second  Third
0  0.666667     0.5    3.0
1  4.000000     3.5    7.0
2  0.888889     2.0    0.5
3       NaN     NaN    NaN
```

If we do not want a **NaN** signifying missing values to be imputed, we can use the **fill_value** attribute to substitute with a default value. However, to take advantage of the **fill_value** attribute, we have to use the Pandas arithmetic methods: **add(), sub(), mul(),**

div(), floordiv(), mod(), and **pow()** for addition, subtraction, multiplication, integer division, numeric division, remainder division, and exponentiation. Let's see examples.

```
df_A.add(df_B, fill_value=10)
'Output':
   First  Second  Third
0    5.0     9.0   12.0
1   10.0     9.0    8.0
2   17.0     9.0   12.0
3   12.0    19.0   12.0
```

Combining Datasets

We may need to combine two or more datasets together; Pandas provides methods for such operations. We would consider the simple case of combining data frames with shared column names using the **concat** method.

```
# combine two dataframes column-wise
pd.concat([df_A, df_B])
'Output':
   First  Second  Third
0      2       3      9
1      8       7      7
2      8       6      4
0      3       6      3
1      2       2      1
2      9       3      8
3      2       9      2
```

Observe that the **concat** method preserves indices by default. We can also concatenate or combine two dataframes by rows (or horizontally). This is done by setting the **axis** parameter to 1.

```
# combine two dataframes horizontally
pd.concat([df_A, df_B], axis=1)
```