```
my_DF.isnull()
'Output':
     age  state_of_origin
0  False            False
1  False            False
2   True            False
3  False            False
4  False             True
```

However, if we want a single answer (i.e., either **True** or **False**) to report if there is a missing data in the data frame, we will first convert the DataFrame to a NumPy array and use the function **any()**.

The **any** function returns **True** when at least one of the elements in the dataset is **True**. In this case, **isnull()** returns a DataFrame of booleans where **True** designates a cell with a missing value.

Let's see how that works.

```
my_DF.isnull().values.any()
'Output':  True
```

Removing Missing Data

Pandas has a function **dropna()** which is used to filter or remove missing data from a DataFrame. **dropna()** returns a new DataFrame without missing data. Let's see examples of how this works.

```
# let's see our dataframe with missing data
my_DF = pd.DataFrame({'age': [15,17,np.nan,29,25], \
            'state_of_origin':['Lagos', 'Cross River', 'Kano',
            'Abia', np.nan]})
my_DF
'Output':
    age state_of_origin
0  15.0           Lagos
1  17.0     Cross River
2   NaN            Kano
3  29.0            Abia
4  25.0             NaN
```

```
# let's run dropna() to remove all rows with missing values
my_DF.dropna()
'Output':
    age state_of_origin
0  15.0           Lagos
1  17.0     Cross River
3  29.0            Abia
```

As we will observe from the preceding code block, **dropna()** drops all rows that contain a missing value. But we may not want that. We may rather, for example, want to drop columns with missing data or drop rows where all the observations are missing or better still remove consequent on the number of observations present in a particular row.

Let's see examples of this option. First let's expand our example dataset.

```
my_DF = pd.DataFrame({'Capital': ['Yola', np.nan, np.nan, 'Port-Harcourt',
                      'Jalingo'],
 'Population': [3178950, np.nan, 2321339, np.nan, 2294800],
 'State': ['Adamawa', np.nan, 'Yobe', np.nan, 'Taraba'],
 'LGAs': [22, np.nan, 17, 23, 16]})
my_DF
'Output':
         Capital  LGAs  Population    State
0           Yola  22.0   3178950.0  Adamawa
1            NaN   NaN         NaN      NaN
2            NaN  17.0   2321339.0     Yobe
3  Port-Harcourt  23.0         NaN      NaN
4        Jalingo  16.0   2294800.0   Taraba
```

Drop columns with **NaN**. This option is not often used in practice.

```
my_DF.dropna(axis=1)
'Output':
Empty DataFrame
Columns: []
Index: [0, 1, 2, 3, 4]
```

Drop rows where all the observations are missing.

```
my_DF.dropna(how='all')
'Output':
        Capital  LGAs  Population    State
0          Yola  22.0   3178950.0  Adamawa
2           NaN  17.0   2321339.0     Yobe
3  Port-Harcourt  23.0         NaN      NaN
4       Jalingo  16.0   2294800.0   Taraba
```

Drop rows based on an observation threshold. By adjusting the **thresh** attribute, we can drop rows where the number of observations in the row is less than the **thresh** value.

```
# drop rows where number of NaN is less than 3
my_DF.dropna(thresh=3)
'Output':
   Capital  LGAs  Population    State
0     Yola  22.0   3178950.0  Adamawa
2      NaN  17.0   2321339.0     Yobe
4  Jalingo  16.0   2294800.0   Taraba
```

Imputing Values into Missing Data

Imputing values as substitutes for missing data is a standard practice in preparing data for machine learning. Pandas has a **fillna()** function for this purpose. A simple approach is to fill **NaNs** with zeros.

```
my_DF.fillna(0) # we can also run my_DF.replace(np.nan, 0)
'Output':
        Capital  LGAs  Population    State
0          Yola  22.0   3178950.0  Adamawa
1             0   0.0         0.0        0
2             0  17.0   2321339.0     Yobe
3  Port-Harcourt  23.0         0.0        0
4       Jalingo  16.0   2294800.0   Taraba
```