python_Numpy_Package

June 12, 2021

1 Python Numpy Package

[2]:
```
# Numpy is a Python package that stands for "numerical Python." It is a library␣
 ↪consisting of multidimensional array objects and a collection of routines␣
 ↪for processing arrays.
#The Numpy library is used to apply the following operations:
#• Operations related to linear algebra and random number generation
#• Mathematical and logical operations on arrays
#• Fourier transforms and routines for shape manipulation
#For instance, you can create arrays and perform various operations such as␣
 ↪adding or subtracting arrays
```

1.1 Addition and subtraction of arrays using np.add and np.subtract

[4]:
```
import numpy as np
#addition of arrays
a=np.array([[1,2,3],[4,5,6]])
b=np.array([[7,8,9],[10,11,12]])
np.add(a,b)
```

[4]:
```
array([[ 8, 10, 12],
       [14, 16, 18]])
```

[5]:
```
#subtraction of arrays
np.subtract(a,b) #Same as a-b
```

[5]:
```
array([[-6, -6, -6],
       [-6, -6, -6]])
```

1.2 Data Cleaning and Manipulation Techniques

[ ]:
```
#Keeping accurate data is highly important for any data scientist. Developing␣
 ↪an accurate model and getting accurate predictions from the applied model␣
 ↪depend on the missing values treatment. Therefore, handling missing data is␣
 ↪important to make models more accurate and valid.
#Numerous techniques and approaches are used to handle missing data such as the␣
 ↪following:
```

```
#• Fill NA forward
#• Fill NA backward
#• Drop missing values
#• Replace missing (or) generic values • Replace NaN with a scalar value
#The following examples are used to handle the missing values in a tabular data␣
 ↪set:
#In [31]: dataset.fillna(0) # Fill missing values with zero value
#In [35]: dataset.fillna(method='pad') # Fill methods Forward
#In [35]: dataset.fillna(method=' bfill') # Fill methods Backward
#In [37]: dataset.dropna() # remove all missing data
```

1.3 Abstraction of the Series and Data Frame

[7]:
```
# A series is one of the main data structures in Pandas. It differs from lists␣
 ↪and dictionaries. An easy way to visualize this is as two columns of data.␣
 ↪The first is the special index, a lot like the dictionary keys, while the␣
 ↪second is your actual data.
```

1.3.1 Create a series

[8]:
```
import pandas as pd
animals = ["Lion", "Tiger", "Bear"]
pd.Series(animals)
```

[8]:
```
0     Lion
1    Tiger
2     Bear
dtype: object
```

[9]:
```
marks = [95, 84, 55, 75]
pd.Series(marks)
```

[9]:
```
0    95
1    84
2    55
3    75
dtype: int64
```

[10]:
```
quiz1 = {"Robert":75, "Lillian": 84, "Jackie": 70}
q = pd.Series(quiz1)
q
```

[10]:
```
Robert     75
Lillian    84
Jackie     70
dtype: int64
```

1.3.2 query a series using a series using iloc(index) or loc(label)

loc()

```
[15]: q.loc['Robert']
```

```
[15]: 75
```

```
[16]: q['Robert']
```

```
[16]: 75
```

iloc()

```
[17]: q.iloc[2]
```

```
[17]: 70
```

```
[18]: q.iloc[2]
```

```
[18]: 70
```

1.4 Numpy operation on a series

```
[19]: import pandas as pd
      import numpy as np
      s = pd.Series([70,90,65,25, 99])
      s
```

```
[19]: 0    70
      1    90
      2    65
      3    25
      4    99
      dtype: int64
```

1.4.1 summation using a loop

```
[20]: total =0
      for val in s:
          total += val
      print (total)
```

```
349
```

1.4.2 Summation using Numpy (faster function)

```
[21]: total = np.sum(s)
      print (total)
```

```
349
```

1.4.3 Alter a series to add new values

[22]:
```
s = pd.Series ([99,55,66,88])
s.loc['Robert'] = 85
s
```

[22]:
```
0          99
1          55
2          66
3          88
Robert     85
dtype: int64
```

1.4.4 append two or more series

[25]:
```
test = [95, 84, 55, 75] #create list
marks = pd.Series(test) #create serie 1 out of list
s = pd.Series ([99,55,66,88]) #create serie 2
s.loc['Ahmed'] = 85 #add value to serie 2
NewSeries = s.append(marks) #append seri1 to serie 2
NewSeries #print final serie
```

[25]:
```
0         99
1         55
2         66
3         88
Ahmed     85
0         95
1         84
2         55
3         75
dtype: int64
```