Popular Packages: Examples

When I talk about popular packages in Python, it includes both built-in and third-party libraries. Once imported within the program, the usage of these packages follows the same structure and rules as regular code you would encounter without the import. You have explored some of the popular package names in the domains of data science, ML, and Web earlier on in the course. Here are a few examples of using these packages that will help you get comfortable with the idea.

Before you use any package, the first piece of code that you must always use is the **`import`** statement. That is true even in the case of built-in packages. For example, if you want to use the json package, you will first add a line such as:

```
import json

```

Numpy

Assuming there is already an installation for the numpy package, the code for it can be as follows:

```
import numpy as np

a = np.zeros(10)
print(a)

b = np.full((2,10), 0.7)
print(b)

c = np.linspace(0,25,7)
print(c)

print(type(c))

```

The output for the code above is:

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[[0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7]
 [0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7]]
[ 0.          4.16666667  8.33333333 12.5        16.66666667 20.83333333  25.        ]
<class 'numpy.ndarray'>

```

The **`zeros()`** function inside numpy creates an array with **`n`** number of zeroes inside it.

The **`full()`** function creates a two-dimensional matrix of dimensions 2 x 10 consisting only of the values 0.7.

In the example, **`linspace()`** function divides the values between 0 and 25 in 7 equal parts. The resultant matrix is in the output.

Finally, when you see the data type of c, it is a special data-type created and used in numpy called as **ndarray**. If you try the output for **`a`** and **`b`**, it will also be **ndarray** as numpy deals exclusively with **ndarray**, which is a substitute for lists and is far more efficient.

These are some of the functions provided by numpy.

Pandas

Now you will explore the usage of another library that closely works with numpy and other data science libraries called pandas.

```
import pandas as pd

a = pd.DataFrame({'Animals': ['Dog','Cat','Lion','Cow','Elephant'],
```

```
                    'Sounds':['Barks','Meow','Roars','Moo','Trumpet']})

print(a)
print(a.describe())

b = pd.DataFrame({
    "Letters" : ['a', 'b', 'c', 'd', 'e', 'f'],
    "Numbers" : [12, 7, 9, 3, 5, 1]  })

print(b.sort_values(by="Numbers"))

b = b.assign(new_values = b['Numbers']*3)
print(b)


```

Output:

```
   Animals   Sounds
0      Dog    Barks
1      Cat     Meow
2     Lion    Roars
3      Cow      Moo
4 Elephant  Trumpet


       Animals Sounds
count        5      5
unique       5      5
top        Dog  Barks
freq         1      1


  Letters  Numbers
5       f        1
3       d        3
4       e        5
1       b        7
2       c        9
0       a       12


  Letters  Numbers  new_values
0       a       12          36
1       b        7          21
2       c        9          27
3       d        3           9
4       e        5          15
5       f        1           3

```

In the four outputs in this code, I created a pandas DataFrame in the code above called **a**.

The first output is for the DataFrame called **a** that displays the output in a very systematic format.

The second output uses the **describe()** function in pandas that will give the count, frequency, top values and frequency among other values.
In the second DataFrame, **b** consists of letters and numbers in random order.

The third output is a sorting function that will provide a sorted table leading to shuffling of the data entries in the table.

Lastly, the **`assign()`** function takes the values present inside the table, performs an operation over them and creates a new variable called **`new_values`** that is then added to the table.

Pandas, just like Numpy is very widely used and has a vast variety of functionalities present in addition to the ones mentioned.
NLTK
NLTK as mentioned earlier, is a library in Python used for Natural Language Processing. Here are some of the things you can do with it.

```
1   import nltk
2   from nltk.tokenize import word_tokenize
3   from nltk.corpus import stopwords
4
5   text = "Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has
6
7   # Print statement 1
8   print(word_tokenize(text))
9   # Print statement 2
10  print(nltk.tokenize.sent_tokenize(text))
11
12  stopwords = stopwords.words("english")
13  new_text = []
14  for i in text.split():
15      if i not in stopwords:
16          new_text.append(i)
17
18  # Print statement 3
19  print(new_text)
20
```

Output:

```
1   ['Lorem', 'Ipsum', 'is', 'simply', 'dummy', 'text', 'of', 'the', 'printing', 'and', 'typesetting',
2
3   ['Lorem Ipsum is simply dummy text of the printing and typesetting industry.', "Lorem Ipsum has be
4
5   ['Lorem', 'Ipsum', 'simply', 'dummy', 'text', 'printing', 'typesetting', 'industry.', 'Lorem', 'Ip
6
```

NLTK is a huge library and it is inadvisable to import all its packages and subpackages. If you examine the code, you will realize that only the required functionalities from the subpackages such as **corpus** and **tokenize** are imported within the code.
First a block of text is copied inside the code-block and assigned to a variable called **`text`**.

The first function used is **`word_tokenize().`** This takes this text and produces the first part of the output in which the words are 'tokenized' or simply separated by a whitespace. The same can be done with the **`split()`** function in the string, but the use of the package is far more efficient when it comes to larger blocks of code.
The second function **`sent_tokenize()`** takes this block of text and tokenizes by 'sentences'.

For the third output, I first split the code and remove what is called 'stopwords'. Stopwords are words in the English language that can be considered redundant and adding little value while you are undertaking natural language processing. These include words such as 'a', 'the', 'him'. First I'll create a list of these stopwords and then remove them using a **`for loop`** to form a new list called **`new_text`**. You will notice the difference by comparing the first and the final output of the code.

We have covered only a couple of examples here from couple of libraries and there is a plethora of options available with different packages in Python. The best way to learn them is through practice and exploration.