FAQs - Python for Data Science

1. What is indexing?

Indexing is used to access the items/data stored inside a list, a tuple, a dictionary, a data frame, etc.

```
lst1 = [1,2,4,5] #list
dct1 = {'one': 1, 'two':2, 'three':3} #dictionary
tup1 = (1,2,3,4,5) #tuple
set1 = {1,2,3,4} # set
```

Indexing value '2' from all of the above:

```
lst[1] = 2 # use index position to access the value
dct['two']=2 #use the key name to access the value
tupl[1]=2 # use index position to access the value
#set does not support indexing in python
```

Consider below DataFrame:

```
data = [[1,2],[3,4]] #a nested list (a list containing 2 lists)
df = pd.DataFrame([[1,2],[3,4]], columns=['Col1','Col2'])
```

df

	Col1	Col2
0	1	2
1	3	4

Indexing in Data Frames:

```
df['Col2'] #will return all values in Col2
df['Col2'][0] #will return the value at index 0 of Col2
Using iloc in DataFrame to access values
df.iloc[0,1] #will return the value 2
```

'iloc' has two arguments, the first one is for rows and the second one is for columns. 0 in the above code gets the elements of the 0th index and 1 is for the second column in the data frame df.

```
df.iloc[:,1] #will give all rows of Col2
df.iloc[:,:] #will give all rows and columns
df.iloc[1,:] #will give all columns and second row
```

2. What is the difference between an array and a list in Python?

Both the list and array are similar but one major difference is that an array is optimized for arithmetical operations. (Numerical Python – numpy)

3. The drop() function in pandas doesn't save the new DataFrame with a dropped column?

The drop() function has a parameter 'inplace' which is set to False by default. That means it will not make changes to the original data frame but return a temporary copy of the data frame. You can use inplace = True inside the drop function to retain the changes.

4. I want to group the Age column in my data into intervals of 5 years. How should I go about it?

The cut function in Pandas helps in binning numerical variables. Read more here.

5. How does slicing work in python? What is the result of the slice here-

```
arr = np.array([1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
arr[0:2, 1:4]
```

Please note two important points with in regards to the question-

Python indexes start from 0. That is, the first element of an array (or any other similar container) is at index 0.

The slices are inclusive of the starting index but exclude the ending index. That is, a slice 3:7 includes the 3rd index but excludes the 7th index.

Having, seen the above two points, answer the question.

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[0:2, 1:4])

While slicing, the first slice is for the rows and the second(after the comma) is for the columns.

Let us consider the row slice first. The slice is 0:2. Owing to point 2 above, it includes the 0th index but excludes the 2nd one. Thus, rows with indexes 0 and 1 are selected. Owing to point 1, the first and the second rows are selected

Now, these selected rows are moved for the further slice. Let us take the column slice. 1:4 suggests that the 1st index is included and the 4th is excluded. That is, the second, third, and the fourth row is considered in the sliced rows.

Combining the two slices together, we get to the required output, i.e
[[2,3,4],[7,8,9]]

6. I did not understand the logic of how the linspace command in numpy works. Please help.

To check the functionality and the attributes of a function in python please use python help.

```
?np.linspace
or
help(np.linspace)
```

Let us see how linspace works using some examples -

```
np.linspace(10,20,3)
```

array([10. 15. 20.])

In this case, it returns *3* evenly spaced samples, calculated over the interval *[10, 20]*

```
np.linspace(10,20,4)
```

array([10. , 13.33333333, 16.66666667, 20.])

This returns 4 evenly spaced samples, calculated over the interval [10,20].
The endpoint of the interval can optionally be excluded, by changing the attribute *endpoint*.

```
np.linspace(10,20,4, endpoint = False)
```

array([10. , 12.5, 15. , 17.5])

This returns 4 evenly spaced samples, calculated over the interval [10,20), excluding the endpoint 20.

7. Why is my code not running? What is a syntax error?

```
In [2]: import numpy as np
        import pandas as pd

In [3]: df7=pd.DataFrame({"customer":['101','102','103','104'],
                          'category':['cat2','cat2','cat1','cat3'],
                          'important':['yes','no','yes','yes'],
                          'sales':[123,52,214,663]})
        df8=pd.DataFrame({"customer":['101','103','104','105'],
                          'color':['yellow','green','green','blue'],
                          'distance':[12,9,44,21],
                          'sales':[123,214,663,331]})
          File "<ipython-input-3-01b78bf847d7>", line 1
            df7=pd.DataFrame({"customer":['101','102','103','104'],
                              ^
        SyntaxError: invalid syntax
```

Syntax errors in Python are the most basic type of error. Python must follow strict syntax to compile correctly, any aspects of the code that do not conform to the syntax of the programming language will produce a syntax error.

Unlike logic errors, which are errors in the flow or logic of a program, syntax errors are small grammatical mistakes, sometimes limited to a single character like a colon ":", semi-colon ";", brackets "{" "(" "[" etc.

It looks like you are trying to create a data frame from a dictionary. Please use the following code. Please note the change in the brackets, from (to {.

```
import pandas as pd
df7 = pd.DataFrame({"customer":["101","102","103","104"], "category":["cat2","cat2","cat1","cat3"],
                    "important":["yes","no","yes","yes"],"sales":[123,52,214,663]})
df7
```

The "keys" of the dictionary become the name of the column in the data frame and the "values" of that key, given inside a list, take different values in rows of the data frame.

8. Please explain how the apply() function of the "Pandas" library can be applied to both series and data frames. Further, how can it be applied to return some value after passing each row/column of a data frame with some function(default and user-defined)

I. The apply() function of the "Pandas" library of Python, can be applied to both series and data frames.

Applying "apply()" to series -

```
import pandas as pd
import numpy as np
df = pd.DataFrame([[4, 9],[3,6],[7,8]], columns=['A', 'B'])
print(df)
df['A'].apply(np.sqrt)
```

```
   A  B
0  4  9
1  3  6
2  7  8

Out[1]:

0    2.000000
1    1.732051
2    2.645751
Name: A, dtype: float64
```

Applying "apply()" to data frame -

```
df = pd.DataFrame([[4, 9],[3,6],[7,8]], columns=['A', 'B'])
print(df)
df.apply(np.sqrt)
```

```
   A  B
0  4  9
1  3  6
2  7  8

Out[2]:
```

	A	B
0	2.000000	3.000000
1	1.732051	2.449490
2	2.645751	2.828427

II. The apply() function of the "Pandas" library of Python, returns some value after passing each row/column of a data frame with some function. The function can be both default and user-defined.

Applying apply() on a data frame such that it returns some value after passing each column(default function np.sum)-

```
df.apply(np.sum, axis = 0)
```

```
Out[3]:

A    14
B    23
dtype: int64
```

Applying apply() on a data frame such that it returns some value after passing each row(default function np.sum)-

```
df.apply(np.sum, axis=1)
```

```
Out[4]:

0    13
1    9
2    15
dtype: int64
```

```
Applying apply() on a data frame such that it returns some value after passing each column (user-defined function 'sum_input')-
```

```
def sum_input(s):
    return sum(s)
df.apply(sum_input, axis=0)
```

```
Out[5]:

A    14
B    23
dtype: int64
```

Applying apply() on a data frame such that it returns some value after passing each row (user-defined function 'sum_input')-

```
def sum_input(s):
    return sum(s)
df.apply(sum_input, axis=1)
```

```
Out[6]:

0    13
1    9
2    15
dtype: int64
```

Previous Next