Python Advance Assignment 2

1. Explain three-dimensional data indexing.

Answer:-

NumPy arrays are high-performance data structures, better suited for mathematical operations than Python's native list data type. A three-dimensional (3D) array is composed of 3 nested levels of arrays, one for each dimension.

USE numpy.array() TO CREATE A 3D NUMPY ARRAY WITH SPECIFIC VALUES

Call numpy.array(object) with object as a list containing x nested lists, y nested lists inside each of the x nested lists, and z values inside each of the y nested lists to create a x-by-y-by-z 3D NumPy array.

```
a_3d_array = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print(a_3d_array)
```

OUTPUT

```
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
```

USE numpy.zeros() TO CREATE A 3D NUMPY ARRAY OF ZEROS

Call numpy.zeros(shape) with shape as (x, y, z) to create a x-by-y-by-z 3D NumPy array of zeros.

```
a_3d_array = np.zeros((2, 2, 2))
print(a_3d_array)
```

OUTPUT

```
[[[0. 0.]
```

[0. 0.]]

[[0. 0.]

[0. 0.]]]

Further reading: The NumPy library provides high-performance mathematical functionalities. You can read more about it here.

2. What's the difference between a series and a dataframe?

Answer:-

Series

A **pandas series** is a one-dimensional data structure that comprises of a key-value pair. It is similar to a python dictionary, except it provides more freedom to manipulate and edit the data.

To initialize a series, use `pd.Series()`:

```
    import pandas as pd


##### INTIALIZATION #####

#STRING SERIES
fruits = pd.Series(["apples", "oranges", "bananas"])

print("MY FRUIT SERIES")
print(fruits, "\n")

#FLOAT SERIES
temperature = pd.Series([32.6, 34.1, 28.0, 35.9], index = ["a","b","c","d"])

print("TEMPERATURE IN CELSIUS")
print(temperature, "\n")

#INTEGER SERIES
factors_of_12 = pd.Series([1,2,4,6,12], name = "factors of 12")
```

```
print("FACTORS OF 12 SERIES")
print(factors_of_12, "\n")

##### QUERY #####

#USING INDEX
print ("2nd fruit: ", fruits.iloc[1])
#OR
print ("2nd fruit: ", fruits[1], "\n")

#USING KEY
print ("temperature at key \"b\": ", temperature.loc["b"])
```

DATA FRAMES

Obviously, making your DataFrames is your first step in almost anything that you want to do when it comes to data munging in Python. Sometimes, you will want to start from scratch, but you can also convert other data structures, such as lists or NumPy arrays, to Pandas DataFrames. In this section, you'll only cover the latter. However, if you want to read more on making empty DataFrames that you can fill up with data later, go to **question 7**.

Among the many things that can serve as input to make a 'DataFrame', a NumPy `ndarray` is one of them. To make a data frame from a NumPy array, you can just pass it to the `DataFrame()` function in the `data` argument.

Pay attention to how the code chunks above select elements from the NumPy array to construct the DataFrame: you first select the values that are contained in the lists that start with `Row1` and `Row2`, then you select the index or row numbers `Row1` and `Row2` and then the column names `Col1` and `Col2`.

Next, you also see that, in the DataCamp Light chunk above, you printed out a small selection of the data. This works the same as subsetting 2D NumPy arrays: you first indicate the row that you want to look in for your data, then the column. Don't forget that the indices start at 0! For `data` in the example above, you go and look in the rows at index 1 to end, and you select all elements that come after index 1. As a result, you end up selecting `1`, `2`, `3` and `4`.

This approach to making DataFrames will be the same for all the structures that `DataFrame()` can take on as input.

Try it out in the code chunk below:

Remember that the Pandas library has already been imported for you as `pd`.

Note that the index of your Series (and DataFrame) contains the keys of the original dictionary, but that they are sorted: Belgium will be the index at 0, while the United States will be the index at 3.
After you have created your DataFrame, you might want to know a little bit more about it. You can use the `shape` property or the `len()` function in combination with the `.index` property:

These two options give you slightly different information on your DataFrame: the `shape` property will provide you with the dimensions of your DataFrame. That means that you will get to know the width and the height of your DataFrame. On the other hand, the `len()` function, in combination with the `index` property, will only give you information on the height of your DataFrame.
This all is totally not extraordinary, though, as you explicitly give in the `index` property.
You could also use `df[0].count()` to get to know more about the height of your DataFrame, but this will exclude the `NaN` values (if there are any). That is why calling `.count()` on your DataFrame is not always the better option.
If you want more information on your DataFrame columns, you can always execute `list(my_dataframe.columns.values)`. Try this out for yourself in the DataCamp Light block above!

Fundamental DataFrame Operations

Now that you have put your data in a more convenient Pandas DataFrame structure, it's time to get to the real work!
This first section will guide you through the first steps of working with DataFrames in Python. It will cover the basic operations that you can do on your newly created DataFrame: adding, selecting, deleting, renaming, … You name it!

3. What role does pandas play in data cleaning?

Answer:-

Pandas is one of the most popular Python packages used in data science. Pandas offer a powerful, and flexible data structure (**Dataframe & Series**) to manipulate, and analyze the data. Visualization is the best way to interpret the data.
Python has many popular plotting libraries that make visualization easy. Some of them are **matplotlib**, **seaborn**, and **plotly**. It has great integration with matplotlib. We can plot a dataframe using **the plot()** method. But we need a dataframe to plot. We can create a dataframe by just passing a dictionary to the **DataFrame()** method of the pandas library.
Let's create a simple dataframe:

```
# importing required library
# In case pandas is not installed on your machine
# use the command 'pip install pandas'.
import pandas as pd
import matplotlib.pyplot as plt

# A dictionary which represents data
data_dict = { 'name':['p1','p2','p3','p4','p5','p6'],
                'age':[20,20,21,20,21,20],
                'math_marks':[100,90,91,98,92,95],
                'physics_marks':[90,100,91,92,98,95],
                'chem_marks' :[93,89,99,92,94,92]
                }

# creating a data frame object
df = pd.DataFrame(data_dict)

# show the dataframe
# bydefault head() show
# first five rows from top
df.head()
```

Output:

	name	age	math_marks	physics_marks	chem_marks
0	p1	20	100	90	93
1	p2	20	90	100	89
2	p3	21	91	91	99
3	p4	20	98	92	92
4	p5	21	92	98	94

Plots

There are a number of plots available to interpret the data. Each graph is used for a purpose. Some of the plots are BarPlots, ScatterPlots, and Histograms, etc.

Scatter Plot:

To get the scatterplot of a dataframe all we have to do is to just call the **plot()** method by specifying some parameters.

4. How do you use pandas to make a data frame out of n-dimensional arrays?

Answer:-

Different ways to create Pandas Dataframe

Pandas DataFrame is a 2-dimensional labeled data structure with columns of potentially different types. It is generally the most commonly used pandas object.

Pandas DataFrame can be created in multiple ways. Let's discuss different ways to create a DataFrame one by one.

Method #1: Creating Pandas DataFrame from lists of lists.

```
# Import pandas library

import pandas as pd


# initialize list of lists

data = [['tom', 10], ['nick', 15], ['juli', 14]]


# Create the pandas DataFrame

df = pd.DataFrame(data, columns = ['Name',
'Age'])


# print dataframe.

df
```

out put:-

	Name	Age
0	tom	10
1	nick	15
2	juli	14

Method #2: Creating DataFrame from dict of narray/lists
To create DataFrame from dict of narray/list, all the narray must be of

same length. If index is passed then the length index should be equal to the length of arrays. If no index is passed, then by default, index will be range(n) where n is the array length.

```
# Python code demonstrate creating

# DataFrame from dict narray / lists

# By default addresses.


import pandas as pd


# initialize data of lists.

data = {'Name':['Tom', 'nick', 'krish', 'jack'],

        'Age':[20, 21, 19, 18]}


# Create DataFrame

df = pd.DataFrame(data)


# Print the output.
```

Df

Out put

	Name	Age
0	Tom	20
1	nick	21
2	krish	19
3	jack	18

5. Explain the notion of pandas plotting

Answer:-

Pandas is one of the most popular Python packages used in data science. Pandas offer a powerful, and flexible data structure (**Dataframe & Series**) to manipulate, and analyze the data. Visualization is the best way to interpret the data.
Python has many popular plotting libraries that make visualization easy. Some of them are **matplotlib**, **seaborn**, and **plotly**. It has great integration with matplotlib. We can plot a dataframe using **the plot()** method. But we need a dataframe to plot. We can create a dataframe by just passing a dictionary to the **DataFrame()** method of the pandas library.
Let's create a simple dataframe:

```
# importing required library

# In case pandas is not installed on your machine

# use the command 'pip install pandas'.

import pandas as pd

import matplotlib.pyplot as plt
```

```
# A dictionary which represents data
data_dict = { 'name':['p1','p2','p3','p4','p5','p6'],
              'age':[20,20,21,20,21,20],
              'math_marks':[100,90,91,98,92,95],
              'physics_marks':[90,100,91,92,98,95],
              'chem_marks' :[93,89,99,92,94,92]
              }

# creating a data frame object
df = pd.DataFrame(data_dict)

# show the dataframe
# bydefault head() show
# first five rows from top
df.head()
```

	name	age	math_marks	physics_marks	chem_marks
0	p1	20	100	90	93
1	p2	20	90	100	89
2	p3	21	91	91	99
3	p4	20	98	92	92
4	p5	21	92	98	94

Plots

There are a number of plots available to interpret the data. Each graph is used for a purpose. Some of the plots are BarPlots, ScatterPlots, and Histograms, etc.

Scatter Plot:

To get the scatterplot of a dataframe all we have to do is to just call the **plot()** method by specifying some parameters.

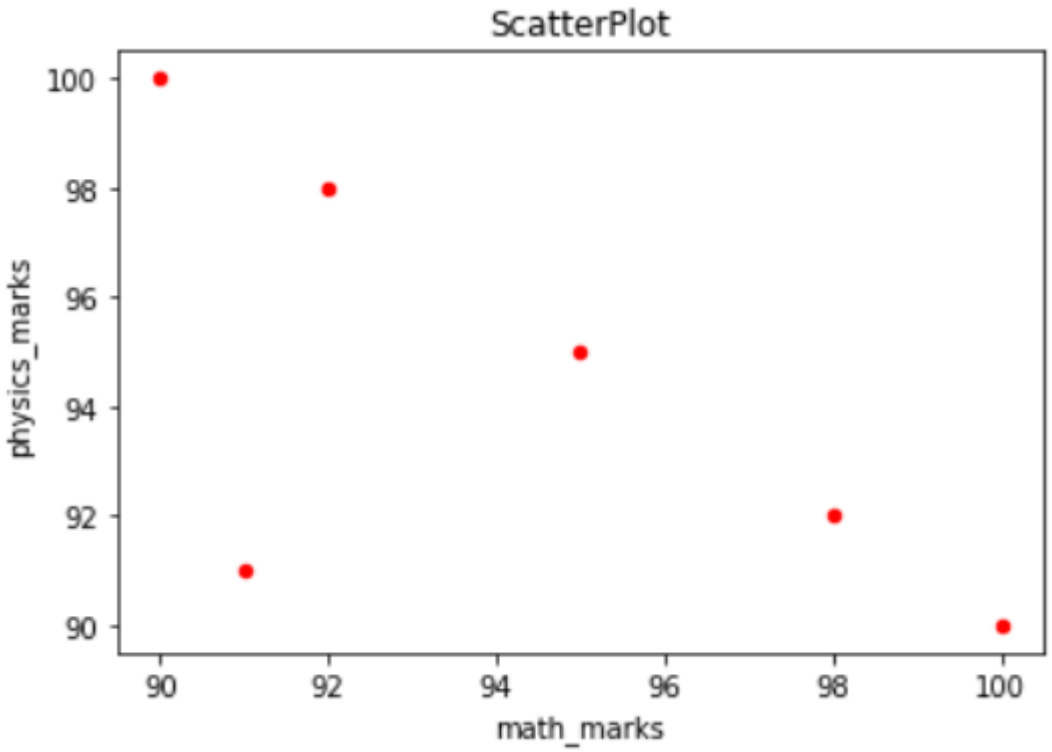
```
# scatter plot
df.plot(kind = 'scatter',
        x = 'math_marks',
        y = 'physics_marks',
        color = 'red')

# set the title
plt.title('ScatterPlot')
```

```
# show the plot

plt.show()
```

out put:-



There are many ways to customize plots this is the basic one.

Bar Plot:
Similarly, we have to specify some parameters for **plot()** method to get the bar plot.

- Python3

```
# bar plot

df.plot(kind = 'bar',

        x = 'name',
```

```
        y = 'physics_marks',
        color = 'green')

# set the title
plt.title('BarPlot')

# show the plot
plt.show()
```