Name: Shubham Shivaji Thotat. Batch:- D9

Page No. Date

Assesement-2 (26/30)

Q.1 Explain the Concept of broad casting in Numpy. Provide an example?

→ i) Broad casting in numpy is a mechanisms that allows arrays with different shapes to be used together in arithmetic operations. When performing operations on arrays of different shapes, Numpy automatically "broadcasts" the arrays to makes their shapes compatible with out the need for explicit compying of data.

ii) This allows for explicit efficient element-wise operations on arrays of different shapes.

Example:

```
import Numpy as np
arr1 = np.array([[1,2,3],
                 [4,5,6],
                 [7,8,9]])
arr2 = np.array([10,20,30])
result = arr1 + arr2
Print("Array 1:")
Print("\n Array 2:")
Print(arr2)
Print("\n Result after broadcasting:")
Print(result)
```

In this example:

i) 'arr1' is a 3x3 array & 'arr2' is a 2x3 array.

ii) Despite having different shapes, Numpy automatically broadcasts 'arr2' to match the shape of 'arr1'.

iii) The values of 'arr2' are extended along the rows to match the shape of 'arr1'.

iv) The addition operation is then performed element-wise betn the two arrays, resulting in a 3x3 array where each element is the sum of the corresponding elements in 'arr1' & 'arr2'.

Broadcasting in Numpy simplifies the syntax & improves the efficiency of operation involving array with different shapes, making it a powerful feature for working with multidimensional arrays.

Q.2. Describe the difference between np.dot() & np.matmul() in Numpy when would you use each function.

→

np.dot() Function	np.matmul() Function
i) The 'np.dot()' Function performs the dot product of two arrays.	i) The 'np.matmul()' Function explicitly performs matrix multiplication.
ii) For 1-D arrays, it performs inner product of vector, For, 2-D arrays it performs matrix multiplication.	ii) It does not perform element-wise multiplication like 'np.dot()' does for 2D arrays.
iii) For N-dimensional arrays, it is a sum product over the last axis of the first array & the second to last axis of the second array.	iii) It can handle higher dimensional array as well, but the behaviour is different compared to 'np.dot()'.
iv) It can also be used to compute the dot product of two vectors.	Code :-

Code :- import numpy as OP.

```
import numpy as np.
A = np.array([[1,2],
              [3,4]]).
B = np.array([[5,6],
              [7,8]]).
```

a = np.array([4,5,6])
b = np.array([4,5,6])
dot_product = np.dot(a,b)

Matrix_product = np.matmul(A,B)

A = np.array([[1,2],
[3,4]])
B = np.array([[5,6],
[7,8]])
Matrix product = np.dot(A,B)

Q.3.

→ a) To display the First 5 rows of Data frame 'sales_Data', you can use the head() method
import pandas as Pd.
print(sales_data.head())
b) To check the display the data types of each column in the Data frame 'sales_data' you can use the 'Info()' method
Print(sales_data.info())

Q.4.

→ c) To calculate the total sales amount for each transaction by adding a new column 'Total_sales', you can simply multiply the 'Quantity Sold' column by hypothetical 'price-per unit' column & assign the result to the new column 'Total_sales'. Assuming 'price_per_unit' is also a column in the DataFrame.
Code:-
Sales_data['Total_sales'] = Sales_data['Quantity_sold']*

Sales-data["Price-Per-unit]
*This will create a new column 'Total sales' in the Dataframe 'sales data' containing the total sales amount for each transaction.

d] To convert the dates in the 'Transaction-Data' column from strings in data time objects for better analysis, you can use 'Pd.to-datetime()' function.
Code:-
Sales-data['Transaction-Date'] = Pd.to-datetime(sales dat['Transaction-Date'])
This will convert the 'Transaction-Date' column from strings to determine objects, allowing for easier manipulation filtering & analysis of dates in the data frame.

Q.5.
→ i) To Find out the average quantity sold per product you can group the data by 'product ID' using the 'groupby()' method & then calculate the mean of the 'Quantity-sold' for each product. Here's how can achieve that;
Code:-
average-quantity per-product = Sales-data groupby('product-ID')['Quantity-sold'].mean()
print(average-quantity-per product)
This will compute the average quantity sold for each product & store the result in a pandas series where the index represents the 'product-ID' & the values represent the average quantity sold you can then use this series for further analysis or visualization.

Page No.
Date

Q.6.
→ a) Numerical Python

Q.7.
→ c) arr = np.array(1,2,3)

Q.8.
→ a) Create an array filled with zeros.

Q.9.
→ a) A two-dimensional labeled data structure

Q.10
→ c) df['Column-name']

Q.11
→ b) Students-data['Age']

Q.12
→ a) Sales-data['price']* Sales-data['Quantity-sold']

Q.13
→ a) Numpy is primarily used for data manipulation & mathematical operations on homogeneous arrays, while pandas provides high level data structures & function to manipulate & analyze structure data like Data frames.

Q.14
→ a) df.iloc[:3]

Q.15
→ a) Drops all rows with missing values

Q.16
→ a) df.apply()

Q.17
→ a) df.sort_values('column_name')

Q.18.
→ b) Returns the largest n values in a specific column

Q.19
→ c) Pdf.to_csv('output.csv')

Q.20
→ b) Converts a column to datatime format.

Q.21
→ a) df.fillna()